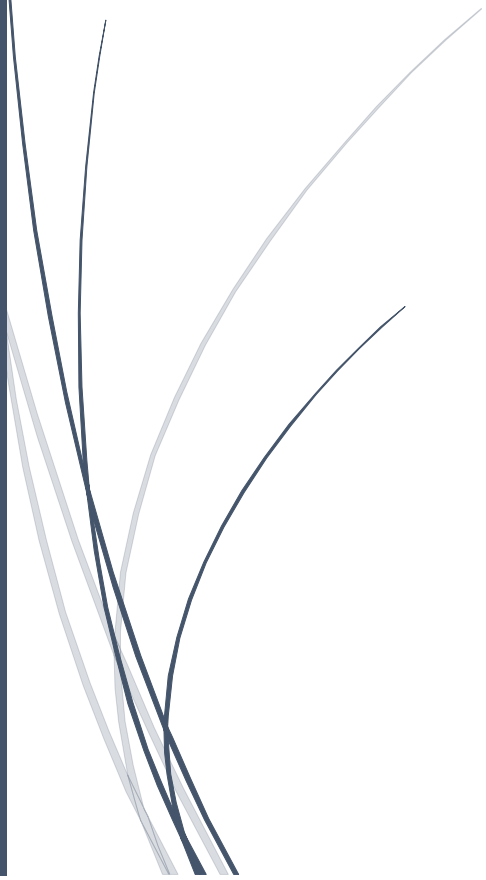
12/14/2018

Introductory Computer Science

Practical Integration of CS Skills for 10-12th Graders in Processing with Python



Ryan Emberling
EDUCATIONAL GOALS, INSTRUCTION, AND ASSESSMENT

Table of Contents

Design Overview

Concept

The seemingly impenetrable abstractness of Computer Science is perhaps the greatest obstacle to fostering interest in the discipline. Students often have difficulty imagining what they could create, which discourages them from engaging. Relatedly, many students of Computer Science have difficulty understanding how the concepts that they learn from lectures can be used to create various effects. Consequently, this course focuses heavily on practice, and using the Processing environment in python mode to facilitate visualizing the core concepts of variables, conditionals, lists, and loops as animation effects directly useful in students' course-long project: their own computer game.

I will tailor my project to 10^{th} - 12^{th} grade learners who wish to take an introductory programming course. Learners are expected to have no programming experience before start of the course, but will have completed Algebra 1. My unit will focus on integration of knowledge and skills that are taught beforehand in the class into students' own personal projects. My structure will expect that before each class period of my unit, students will have spent a corresponding period previously learning the related concepts, so that they are ready to integrate these concepts into their own projects.

Prerequisite Knowledge

	Knowledge	**Skills**	**Dispositions**
Cognitive	Basic Knowledge of: Python Processing library	Computer Literacy Algebraic Literacy Python literacy Processing literacy	Interest in programming

Learning Goals

	Knowledge	**Skills**	**Dispositions**
Cognitive	Concept and uses of: User Input Conditionals Lists For-Loops Functions Classes	Implementing: User Input Conditionals Lists For-Loops Functions Classes	Willingness to fail Desire to create computer programs Inquisitiveness
Meta-Cognitive	What is hard for me? Awareness of own skills	Self-Assessment of code quality Identify possible improvements to their work	Desire to improve own work Interest in creating software

Standards

Design and Development

3-5.AP.11: Create programs that use variables to store and modify data.

3-5.AP.12: Create programs that include events, loops, and conditionals.

Iteration and Improvement

9-12.AP.20: Iteratively evaluate and refine a computational artifact to enhance its performance, reliability, usability, and accessibility.

3-5.AP.14: Create programs by incorporating smaller portions of existing programs, to develop something new or add more advanced features.

Justification

It is vital that students perceive the data structures and tools that they learn in computer science as tools to be used, rather than as abstract concepts distanced from students' own designs. The power and joy of computer science comes from the creative possibility to translate one's imagination into a usable program. This translation requires the ability to apply skills and techniques (like conditionals) to represent relationships. Likewise, computer science is a fundamentally iterative discipline, and it is important that students learn the value of improving upon their designs to add functionality, or refine their approaches for performance and clarity. These two dimensions collectively empower students of computer science to dream up their own creations and turn them into reality.

The key to tapping learner motivation will be allowing students appropriate freedom to create a project of their own interest. The unit will be focused on creation of a game for instructional continuity, and care would be taken to ensure that any student's choice of project is sufficiently complex to facilitate the application of the relevant tools and concepts while being achievable in scope. If students see how what they are learning can be directly used to create something of interest to them, then the learning will be grounded in authentic need for the skills being practiced.

Resources

Personal Experience

I have significant experience with programming as a student (4 college level courses) and an amateur enthusiast (5 years of personal projects), and also as a teacher in private, 1-on-1 settings (2 years tutoring experience in CS).

Professionals in the CS Space

David Kosbie is a CS professor at CMU who is highly regarded for his teaching of introductory CS. He suggested that for the scope of this course the dispositional goal CD0 should take complete primacy over all others, and that it was unlikely that any goals targeting knowledge and skills pertaining to functions and classes could be fruitfully developed in the time in which the course takes place. As a result, I changed the targeted standards to the ones currently included in the course, in order to shift the course focus to fostering student interest in CS through empowering students. Secondly, I switched the course content to cover drawing in processing and using variables instead of functions and classes, to better align with what students in this context can effectively learn through the scope of the course.

Ashley Patton is the Director of Engagement and Outreach in CS at CMU. I contacted her to inquire about a Summer CS course similar in scope to my project. We discussed her passion for CS outreach, and how initiatives that provide CS skills to under-privileged youth can serve as a force for social equity.

She offered to put me in touch with John Balash from ETC and to send me syllabi for the summer outreach programs for comparison.

Articles

https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1006023

- 10 Tips for teaching introductory CS
- Tips are brief, not backed by research, but intuitively sensible (mostly)

https://en.wikiversity.org/wiki/Introduction_to_Programming/About_Programming

- Brief summary of what programming is
- Very dry, not intended to fit into curriculum

https://blogs.edweek.org/edweek/DigitalEducation/2017/05/emerging_research_k-12_computer_science_for_all.html

- Interesting summary of research on key decision points and trends
- Describes inclusiveness as relates to women, minorities, and students with disabilities
- Assessment strategy: be diverse, include more than coding

https://k12cs.org/

- CS Curriculum for k12 with detailed breakdown of high-level goals
- Of the five core concepts, this course will address:
    - Algorithms and Programming
        - Within this concept, this course aims to foster understanding of the concepts of variables and control at the level somewhere in between what this framework labels as “By the end of Grade 8” and “By the End of Grade 12”
- Of the 7 core practices, this course will address:
    - Creating Computational Artifacts
        - This course will address the sub process “Modify an Existing Artifact to Customize or Improve It” as one part of the goals, assessment and instruction.
        - The other sub processes regarding planning, and addressing practical intent, personal expression, or a societal issue are not focuses of this course.

Step A: Learners in Context

Target Population

My unit is designed for students about to begin 10^{th}-12^{th} grade who have grown up using computers and the internet and who have interest in learning to program computers. The program will be situated in a suburban context. Students take this course as an extra-curricular introduction to prepare them for computer science classes and independent projects.

Inquisitiveness and self-evaluation will be encouraged. The material will be new to all the students, and computer science is notoriously intimidating, so it is important that it be made clear to students that not knowing what something is or how to do something will be a normal part of the process, to be

encouraged. Integrating metacognitive assessment of what students have difficulty with as a norm and core task will be useful in helping students to feel they belong even when they are confused.

Essential Questions

Design & Development: 9-12.AP.15

What can I make with Computer Science?

How do CS tools help me make things?

Iteration & Improvement: 9-12.AP.20

What makes a program good?

How could my work be better?

Location

Students will learn outside of school at a tutoring facility. Students will attend group classes in person in the center, and are generally welcome to spend time in the center outside of class to socialize, do work, or ask for help with their personal projects from available tutors, or other students.

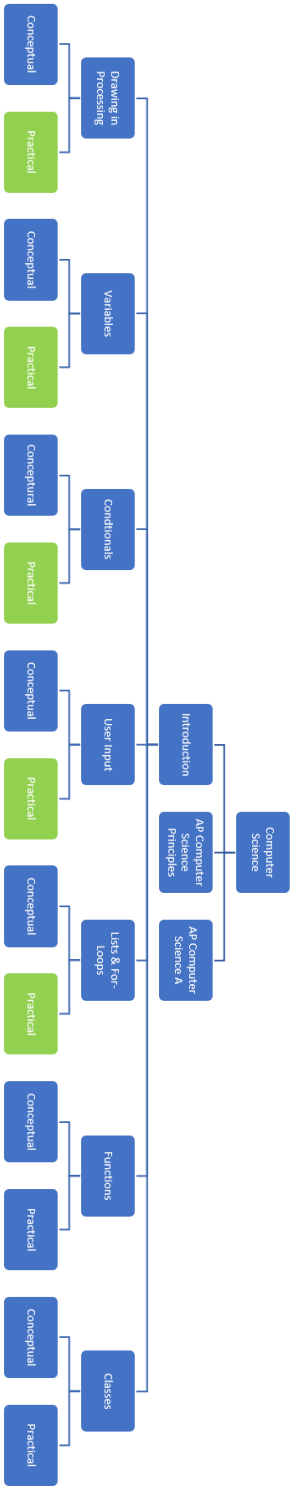
Context within CS Education

The course will take place over the course of one intensive week over the summer. For each of the units, students will begin with a 30-minute warm up "Spiral Review Practice" where they implement all the tools they have learned so far in a small demo. Then they will have a 15-minute conceptual review and lesson on new content, followed by a 15-minute "Isolated Implementation" where they create a demo that showcases the new tool. The students will get a 15-minute break, and then 45 minutes dedicated to practice of implementation of the concepts covered in the previous hour within the context of the students' individual projects (Integrated Implementation).

For each topic, this two-fold (conceptual then practical) lesson plan will be repeated in two lessons per week. Each lesson will be covered for one week, across two lessons, each of which contains a conceptual, then practical component.

This project's scope is a subset of the practical lessons, highlighted in green in the below diagram. It includes the practical labs for drawing, variables, conditionals, user input, and lists with for-loops

Context & Scope Diagram



Program role within Broader Context

This program will prepare students to take college level or AP Computer Science courses with a strong foundational skillset. It is intended to empower students to make use of the quintessential programming tools to create programs of their own design. Students will become familiar with these tools in ad-hoc isolated applications, as well as within the context of a more complex project of their choosing. This is intended to foster their personal interest in Computer Science, and to ground their conceptual understandings in practical applications. Students pursuing further Computer Science education will use all of the skills learned in this program in progressively more complex applications. This may entail learning how to implement these same techniques in other programing languages. The experiences provided by this course are intended to make future learning of these tools in varied contexts come more easily to students, with deeper understanding of the roles played by the constituent parts of their programs. Regardless of whether students pursue further formal education, this course is intended to enable students to pursue projects of their own design.

Developmental Levels

Developmental Level:	Prior Experience:	Individual Differences:
Physical: Students are expected to have the motor skills required for advanced computer use. Social: Students are able to work and converse in groups. Cognitive: Students are able to represent situations with linear and quadratic equations, and to solve these kinds of equations given calculation tools. Metacognitive: Students are expected to have a range of metacognitive abilities upon entering the course, with many students unable to perform accurate self-assessment, and some students able to identify the merits and faults of their work, as well as their own strengths and weaknesses.	Knowledge: Students are expected to know the basics of computer use: how files are organized in folders, how to download and run applications, how to find information on the internet, and how to type. Skills: Students are expected to be able to read, type, solve algebra equations, and model situations with linear and quadratic equations. Students should be able to plot points on a cartesian grid using coordinates. Students are expected to be able to use a computer to find a file within a file structure, download, install and run and application, and organize a folder structure. Dispositions: Students are expected to have interest in learning to program and to create a digital game.	Cognitive: Some students will be better at visualization than others. Likewise, we expect differences in their abstract reasoning capacities. Some students will get to fluidly implementing the tools learned much more quickly than others. Dispositions: Some students will have more grit than others. These students will be more inclined to engage with their mistakes and learn from them, while others may feel discouraged by embarrassment when their program does not work as expected. Some students will be more interested and more inquisitive than others.

Step B: Goal Specification

Cognitive Goals

Knowledge	Skills	Dispositions
CK0: Students will be able to explain that a conditional is a tool to make a program do different things in different circumstances, and will be able to identify contexts in which this is useful, such as keeping a moving figure on the screen, or detecting a collision between figures. CK1: Students will be able to explain that user input is when someone using a computer input does something the program understands, like pressing a key, or clicking the mouse. CK2: Students will be able to explain that a list is a variable that stores multiple values in it, and identify contexts in which lists are useful, such as tracking the position of several similar shapes on screen, or storing all the possible values a particular variable is allowed to hold. CK3: Students will be able to explain that a for-loop is a way of repeating a task once for every element in a list, and be able to identify contexts in which for-loops are useful, such as rendering multiple similar figures in different places on the screen. CK4: Students will be able to explain that a function is a way of grouping related steps together to complete a single task, and be able to identify appropriate contexts in which to define functions, such as drawing a composite figure, or detecting collisions between figures. (Accelerated students only) CK5: Students will be able to explain that a class is a way of grouping different variables and functions together to represent one piece of a program, and be able to identify contexts in which it is helpful to define a class, such as consolidating all code pertinent to a bouncing ball. (Accelerated students only)	CS0: Given a shape or figure that is animated with variables inside processing's draw loop, students will be able to implement conditionals to alter the movement of the shape or figure (e.g. to keep it on-screen) by recognizing that a conditional is needed, writing out the conditions informally, writing out the behaviors informally, writing the conditions in code, and then writing the behaviors in code. CS1: Students will be able to implement for-loops to draw multiple isomorphic figures in processing by identifying the need for a for-loop, informally defining what the program should do during each iteration, writing the first line of the loop in code, and then writing the behavior of the loop in code. CS2: Students will create programs that respond to users clicking the mouse or pressing a key on the keyboard in a way that is sensitive to where the user clicked or which key was pressed by defining mousePressed(), and keyPressed() functions and using the "key", "keyCode", "mouseX", and "mouseY" variables in combinations with conditionals to animate figures on screen in response to user input. CS3: Students will be able to define and call their own functions to perform common discrete tasks including displaying and animating complex figures in Processing by informally writing what the function needs to achieve, defining the function with a name, and writing the individual steps it must take in code. (Accelerated students only) CS4: Students will be able to define and instantiate classes to track, display, and animate objects in Processing by informally defining the behaviors that members of the class will do, informally defining the variables that will be needed to track these behaviors, informally defining the methods that will be needed to implement these behaviors, then defining the class with variables, methods, and a constructor. (Accelerated students only) CS5: Students will be able to draw rectangles, circles, and lines in processing in any size, anywhere on the screen, and filled with any color, by defining the draw() and setup() functions, moving into the draw loop, calling the fill() function to det the color, then calling the appropriate shape drawing function with the correct inputs. CS6: Students will be able to implement variables to draw and animate a composite figure consisting of several shapes by defining global variables in the setup function, globally referencing those variables in the draw function, passing those variables as arguments to several shape drawing functions, then adjusting those variables in the draw function, or user input handling functions, to move the figure to a new location on screen.	CD0: Students will view failure as an assessment of an artifact, not a reflection of their worth, and will subsequently seek to learn from their failures. CD1: Students will develop an interest in creating their own computer programs. CD2: Students will be inquisitive about computer programming

Meta-Cognitive Goals

MK0: Students will be able to identify which concepts and skills are difficult for them by discussing their failures with their peers and the instructor and by comparing their intent to their approach and results. MK1: Students will be able to describe what they are able to create with computer programs by referring to things they have already built and extrapolating about what could be built with similar tools	MS0: Students will be able to accurately assess the quality of their work in terms of its efficiency, clarity, its fulfilment of its intended purpose by comparing their results to their expectations, identifying repetition in their code, and by having their work interpreted by their peers. MS1: Students will be able to identify means to improve their programs by adding features by creatively imagining things that would be engaging or useful to have in a pre-existing program. MS2: Students will be able to identify means to improve their programs by refactoring by writing comments to explain what their code does, replacing unneeded repetition with loops (or functions/classes for advanced students), and re-ordering code to group similar steps.	MD0: Students will desire to improve their own work MD1: Students will monitor their grit, interest, and inquisitiveness about computer science.

Steps C and D: Assessment and Instruction Design

Instructional Overview

Classroom Climate

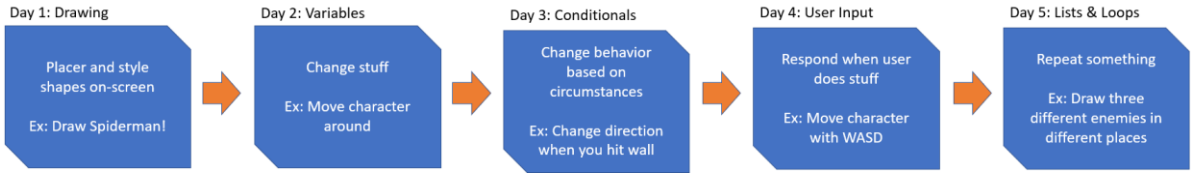
The course will be held with 5-10 enrolled students in a single room of a private tutoring facility, with chairs, a laptop, a projector, and single large table. Each student will additionally provide their own laptop. The instructor will stand at one end of the table, with the projected screen behind her, so she can see students' faces, and that students who wish to look at her while she is talking do not need to move their eyes very far to return their gaze to the projected screen. This will reduce will reduce the cognitive load placed on students by reducing extraneous processing, freeing up capacity to engage with the core issues at hand.

The atmosphere will be welcoming and light-hearted. Students are encouraged to explore, ask questions, and get things wrong without repercussions. Helping each other is encouraged during work time, but students will be expected to type their own code. Talking over other people, including the instructor, will be discouraged, as will any kind of shaming of a student for being wrong or unskilled. These norms will be established in large part by exemplar of the instructor, who should make explicit that everyone who was ever good at anything started as a beginner, that respect is imperative, failure is to be welcomed along with inquisitiveness and fun seeking. The instructor should then follow through by being whimsical and encouraging of struggling students, and by immediately shutting down any negativity if students berate each other. In this way, classroom culture will be guided towards one of open mindedness, acceptance, and trying in the face of potential failure (**Norms[1], Belonging**).

[1] Big ideas referenced from the 2018 EGIA class slab will be inserted parenthetically in bold.

WHERETO

W: A large poster with the units on them, a brief description of what they are, and corresponding functionality that can be built using such a tool (e.g. variables: how you store information- used for controlling where your characters are on screen) will be on the wall of the class. Each class, at the beginning of the Spiral Review Practice activity (See assessments below), the instructor will remind students what they have done so far, where that puts them in the course, what they will build today, and how that could be useful in their projects (**Motivation**). Below is a sample of a poster that could be used to help students track the progression of the course.



H: The course-long project in which every student makes a computer game of their own choosing, as well as the continual opportunities for students to come up with their own figures to draw are intended to foster interest and hook students. In this way, every concept learned will be applied that very same day in order to improve students' games and ground the techniques in an engaging use-case (**Authenticity, Excitement**).

E: Students will be equipped to succeed by being given a conceptual overview and a worked example of every new topic before ingraining the topic with two practice opportunities each class. In addition, a full review of all topics learned heretofore (the Spiral Review Practice) will be held at the beginning of each class to give additional practice and maintain students' understandings over time. Finally, during every performance task, the instructor will walk around and provide scaffolding to students who need it to keep them from getting stuck. This way, all students should be empowered to partake[2] in Making every class (**Making**).

R: During the end of class survey each class, students will be asked how their code could be improved. This will be an opportunity to foster self-reflection so that students can seek to improve their course-project during the subsequent class during the Integrated Implementation. The surveys will serve the role of clarifying to students what is expected of them and fostering a culture of self-reflection (**Transparency, Norms**).

E: The Beginning and End of Course Surveys, as well as the End of Class Survey provide students the opportunities to assess the quality of their work, as well as their interests, and difficulties (**Transparency, Norms**).

T: The inclusion of additional goals involving functions and classes (CK4, CK5, CS3, CS4) allow the instructor to push advanced students to greater achievements than is expected of most students. Additionally, the project time in every class (the Integrated Application) can be used to help struggling students review material they have not mastered, by having them work on these elements of their project. This may result in students ending the course with a less-complete game than others, but this spectrum empowers instructors to let students work at their own pace (**Adaptive Goals**).

[2] ABC's 193

O: The schedule of the course, as shown below, demonstrates how organization is used to ensure that students get ample conceptual overview, practice, and review of each tool throughout the course, and that the goals of the course are instructed and assessed. It also enables the instructor to assess student progress in many different dimensions many times throughout the week-long course (**Alignment**, **Evaluation Research, Triangulation**).

Assessment Overview

Course assessments include informal evidence, two kinds of self-assessment surveys, and three performance tasks. The breadth provided by these assessment forms is intended to align assessment tasks to goals and ensure validity through triangulation (**Alignment, Validity, and Triangulation**). Of the self-assessment surveys, one will be given at the beginning and end of the course to assess longer-term and general dispositional goals, while the other will be given at the end of each class to assess students' abilities to engage with the day's material and to reflect on their understanding.

The three performance assessment tasks are each given once per class, and each is an opportunity to build[3] or expand a program to foster interest and sense-making (**Making**). The Spiral Review Practice involves building a program that incorporates all the tools learned heretofore, and will be done towards the beginning of class. The Isolated Implementation uses worked examples[4] to empower students to engage in deliberate practice[5] of a discrete skill. The Integrated Implementation involves incorporating the technique learned in the Isolated Implementation into the students' course-long project to create a computer game. This makes use of an authentic whole-task to foster interest and directly relate the tools learned in this course to useful and appealing applications (see Authenticity).

Schedule

Day	Time	Instruction	Performance Assessment	Other Assessments
Day 1	15 minutes			Beginning of Course Survey (CD0, CD1, CD2, MD0)
	15 minutes	Conceptual Explanation of New Topic: Drawing in Processing (Beyond the scope of this report)		
	15 minutes	Worked Example for use of new Topic: Drawing in Processing (CS5)		
	15 minutes	Informal/Conversational Support (CS5)	Isolated Implementation (MD0, MK0, CS5, CD0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	45 minutes	Informal/Conversational Support (CS5)	Integrated Implementation (CD0, MD0, CD1, MK0, CS5)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	5 minutes			End of Class Survey (MK0, MD0, MS2)

[3] ABC's 193
[4] ABC's 303
[5] ABC's 50

Day 2	5 minutes	Orient class to what's been covered (Drawing), and today's focus: Variables		
	25 minutes		Spiral Review Practice: Drawing (MK0, MK1, CD0, CS5)	
	15 minutes	Conceptual Explanation of New Topic: Variables (Beyond the scope of this report)		
	15 minutes	Worked Example for use of new Topic (CS6)		
	15 minutes	Informal/Conversational Support (CS5, CS6)	Isolated Implementation (MK0, MD0, CS6)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	45 minutes	Informal/Conversational Support (CS5, CS6)	Integrated Implementation (MK0, MD0, CD0, CD1)	Conversational Assessment
	5 minutes			End of Class Survey (MK0, MD0, MS2)
Day 3	5 minutes	Orient class to what's been covered (Drawing and Variables), and today's focus: Conditionals		
	25 minutes	Informal/Conversational Support (CS5, CS6)	Spiral Review Practice: Drawing and Variables (MK0, MK1, CS5, CD0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	15 minutes	Conceptual Explanation of New Topic: Conditionals (Beyond the scope of this report)		
	15 minutes	Worked Example for use of new Topic (CK0, CS0)		
	15 minutes	Informal/Conversational Support (CK0, CS5, CS6, CS0)	Isolated Implementation (MK0, MD0, CS5, CS6, CS0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	15 minutes	Informal/Conversational Support (CK0, CS5, CS6, CS0)	Integrated Implementation (MK0, MD0, CK0, CS0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	5 minutes			End of Class Survey (MK0, MD0, MS2, CK0)
Day 4	5 minutes	Orient class to what's been covered (Drawing, Variables, and Conditionals), and today's focus: User Input		

	25 minutes	Informal/Conversational Support (CK0, CS5, CS6, CS0)	Spiral Review Practice: Drawing, Variables, and Conditionals (MK0, MK1, CS5, CS6, CS0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	15 minutes	Conceptual Explanation of New Topic: User Input (Beyond the scope of this report)		
	15 minutes	Worked Example for use of new Topic: User Input (CS2)		
	15 minutes	Informal/Conversational Support (CK0, CK1, CS5, CS6, CS2, CS0)	Isolated Implementation (MK0, MD0, CK1, CS0, CS2, CS5, CS6, CD0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	45 minutes	Informal/Conversational Support (CK0, CK1, CS5, CS6, CS2, CS0)	Integrated Implementation (MK0, CD0, CS2)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	5 minutes			End of Class Survey (MK0, MD0, MS2, CK1)
Day 5	5 minutes	Orient class to what's been covered (Drawing, Variables, and Conditionals), and today's focus: Lists and Loops		
	25 minutes		Spiral Review Practice: Drawing, Variables, Conditionals, Lists, and User Input (MK0, MK1, CS5, CS6, CS0, CK1, CS2)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	15 minutes	Conceptual Explanation of New Topic: Lists and For-Loops (Beyond the scope of this report)		
	15 minutes	Worked Example for use of new Topic (CK2, CK3, CS1)		
	15 minutes	Informal/Conversational Support (CK0, CK1, CK2, CK3, CS5, CS6, CS2, CS0)	Isolated Implementation (MK0, CK2, CK3, CS1, CD0)	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	45 minutes	Informal/Conversational Support (CK0, CK1, CK2, CK3, CS5, CS6, CS2, CS0)	Integrated Implementation	Conversational Assessment (MK1, MS0, MS1, MS2, MD1, CD1, CD2)
	5 minutes			End of Class Survey (MK0, MD0, MS2, CK2, CK3)

	15 minutes			End of Course Survey (CD0, CD1, CD2, MD0)

The conceptual introductions to new content, colored in purple in the above schedule, are beyond the scope of this report.

The remainder of this section describes the instructional and assessment activities in greater detail. As each day follows essentially the form, examples will be provided only for Day 4, centered around the introduction of User Input. The class activities are described in the order in which they appear in the course of the day's class. The beginning and end of course survey is only given on Day 1 and Day 5, and it is detailed after the End of Class Survey.

Performance Tasks

The course includes three kinds of performance tasks throughout the course, all of which involve implementing programming tools to achieve a purpose and all of which are performed *each class*. These tasks are designed to ground the abstract concepts and practices students are learning in their own experience.

- **Spiral Review Practice**: Students come up with a new idea for a program that implements all of the skills they have learned so far, and will then implement it in processing.
- **Isolated Implementation**: Students implement a new tool in relative isolation to create a small program that uses the tool to create a visual effect. For example, students could implement detecting arrow key presses to move a shape of their choosing across the screen.
- **Integrated Implementation**: Students incorporate the tool in the context of their long-term project (a game of their choice), to add a new feature required in the game. For example, if building a platformer (a Mario-like game where the primary objective is to move your character to a desired place), students would implement detecting arrow key presses to move their character around in their game.

These assessments are intended to leverage Observation[6] (through the worked examples) and Participation[7] (through building a program), and Making to foster student understanding (**Making**). The use of the Spiral Review Practice at the beginning of each class is intended to provide spaced practice. The pairing of isolated and integrated application tasks is intended to facilitate vertical transfer by gradually deepening the contexts in which students apply computer science tools (**Transfer**).

This course offers no certification, and consequently all assessments are taken to be formative in the sense of solely serving the purpose of facilitating more effective instruction. For this reason, the instructor is expected to wander around the room during all performance tasks every class, and discuss with individual students (see Informal evidence above) to assess their understanding and to give appropriate scaffolding to keep them from remaining stuck.

[6] ABC's of How We Learn pg 180
[7] ABC's of How We Learn pg 193

Spiral Review Practice: Implementing previous tools: Drawing, Variables, and Conditionals

Relevant goals: CS0, CS6, CS5, CD0, MK0, MK1

The class will work together to produce a worked example that applies drawing and conditionals to animate a complex figure on the screen, then each student will individually create their own variation on the worked example to reinforce their understanding with practice (**Worked Example, Making**). Below is the example from Day 4, which gives students the opportunity to practice and review the previous topics of drawing, variables and conditionals.

Instruction

The instructor will ask the class what is needed in order to draw a ball that bounces left and right on the screen, without traveling offscreen. The instructor will then lead a brainstorming session where students name the different things that will need to be included in order to make the program function properly (a setup function, a draw function, variables, a conditional, etc.). The instructor will write these ideas down on a board for everyone to see while the students brainstorm. When the students suggest something that requires further clarification, the instructor will ask for it (e.g., student: "We need a setup function", instructor: "what should go inside the setup function?"). Once a plan has been drafted the instructor will code the example in front of students, asking them along the way what should be typed to implement the solution.

Observations Coded for Cognition

Once the example has been worked out, the instructor will go around the room, and ask everyone to come up with a similar, but different application for these tools, e.g., "I want my ball to pass through and come out the other side when it hits the edge of the screen", "I want my ball to appear in a random location when it crosses the edge of the screen", etc. If students cannot come up with a new variation, the instructor will ask probing questions that lead the student towards a new variation, and if the student still cannot imagine a variation, the instructor will provide one for them. Then the instructor will ask students to implement their variation on the worked example. Likewise, the instructor will ask each student to pick a composite figure consisting of at least two shapes to animate, "circle in a square," "triangle on a rectangle," etc. to draw instead of a ball. The instructor will leave the worked example projected on the screen, and will wander the room, checking in on students, and helping where needed.

Rubric – Spiral Review Practice

Student is able to come up with a novel variation on the bouncing ball (MK1)	1: Student must be told a variation to implement	2: Student comes up with a variation after scaffolding questions	3: Student suggests original variation on the worked example without assistance.
Student is able to come up with their own composite figure to draw (MK1)	1: Student must be told a figure to implement	2: Student comes up with a composite figure to draw after scaffolding questions	3: Student suggests original composite figure without assistance.
When asking for help, student can articulate what they are trying to achieve (intent), how they have attempted it (methods), and what	1: Student can only explain one or none of their intent, methods, results.	2: Student accurately describes 2/3 out of their intent, their methods, and their results.	3: Student accurately describes their intent, methods, and results.

has happened (results) (MK0)			
Student is unashamed to ask for help (CD0)	1: Student does not ask for help when stuck	2: Student asks for help, but expresses signs of embarrassment in tone, body language, diction, etc.	3: Student will ask for help when struggling, without showing signs of embarrassment.
Student is able to create a setup() and draw() functions to prepare their program for complex drawings. (CS5)	1: Student does not produce working setup or draw functions without help	2: Student correctly defines either setup() or draw(), but not both, without help	3: Student correctly defines setup() and draw() functions unassisted
Student initializes global variables for position in setup() function and refers to them globally in the draw() function (CS6)	1: Student does initialize global variables in setup() without help	2: Student initializes global variables in setup() but fails to refer to them globally in draw() without assistance	3: Student correctly initializes global variables in setup() and refers to them in draw()
Student draws a composite figure in a position specified by variable(s) (CS5, CS6)	1: Student fails to draw a complex figure a ball unassisted	2: Student draws composite figure, but fails to position it with variables unassisted	3: Student successfully composite figure with variable positioning without assistance
Student animates their figure by changing the value of the positional variable(s) (CS6)	1: Student requires assistance to animate the ball at all.	2: Student animates ball, but not in the desired way (ball moves incorrectly, or many copies of the ball appear), without assistance	3: Student successfully animates ball in the correct way unassisted
Student implements conditionals to control animation of the figure as described by their variation (CS0)	1: Student does not create any conditional behavior without assistance	2: Student implements some conditional behavior unassisted, but needs assistance to finish implementation.	3: Student implements correct animation for their variation with conditionals unassisted.
Student implements custom function to consolidate their code [optional] (CS4)	1: Student does not define or call their own function	2: Student either defines or identifies where to call their own function, but needs help finishing implementation.	3: Students defines and calls their own function unassisted.
Student implements custom class to consolidate their code [optional] (CS4)	1: Student does not define or instantiate their own class	2: Student either defines or instantiates their own class, but needs help finishing implementation.	3: Students defines and instantiates their own class unassisted.

Isolated Implementation: Handling User Input (Example)

Relevant Goals: CD0, MD0, CK1, MK0, CS0, CS2, CS5, CS6

Students will create a processing program moves an object up, down, left, or right, when the user presses a corresponding key (WSAD). Students will receive a brief conceptual overview of what user input is, and how it can be useful in programming (this overview is out of the scope of this report). Students then observe a worked example from the instructor, demonstrating a use-case for handling user input, and how user to make a program respond to user input. As with the Spiral Review Practice, students will then construct their own variation on the worked example to reinforce their understanding with practice (**Authenticity, Making, Worked Example**). Below is an example from Day 4, focused on creating simple examples that create varying effects by handling User Input.

Instruction

Students will receive scaffolding in the form of a worked example that the instructor programs in front of them on a projected screen immediately before they practice. The worked example will involve moving a character upwards every time the user presses the 'w' key.
Here is sample code that could be used to create such a worked example:

```
def setup():
        size(1000, 1000)
        x = 500
        y = 500
        rectMode(CENTER)
def draw():
        global x, y
        fill(0, 0, 200)
        rect(x, y, 50, 50)
        fill(200, 0, 0)
        ellipse(x, y, 50, 50)
def keyPressed():
        global y
        if key == 'w':
        y -= 10
```

While the instructor is working on parts of the program that the students have previously developed (e.g. setup and draw function definitions, drawing shapes, etc.), she will ask students what is needed

next, and then write the code when a student answers. For example, she might ask the class, “What do we always need when we start writing our processing program?”, expecting the reply “setup and draw functions!” Then she can write the beginning of the setup function and prompt, “And what goes in our setup function?”, waiting for students to suggest calling the size() function, and reminding them to do so if no one replies. This will continue while she defines the contents of the setup and draw functions, until it is time to define the new functionality, keyPressed().

Then she will explain to students that they have already discussed what it means to capture user input, and they have seen a simple example during the conceptual overview. Now she will show them how to create it. She will inform students that processing is on the lookout for a function with the special name keyPressed(), and if you define a function with this name, it will call that function every time you press any key on the keyboard. She will begin writing the function, then remind students that the figure is being positioned based on the variable ‘y’, and that by changing this variable, we can change the position of the figure. She’ll explain that we’ll need to include the line ‘global y’ in order to change this variable inside the keyPressed function. She’ll then change the value of y, when the key is pressed by adding the line, ‘y -= 1’. She will demonstrate the new functionality by running the application and tapping the keyboard to move the figure upwards, hopefully to the amazement of her students.

“But there is a problem with our program. Can anyone tell me why this approach isn’t good enough?” If no one answers, she can prompt further: “What will happen if I press a different key besides ‘w’?” She will then demonstrate by pressing another key, and the class will note that the figure still moves up. “What if I want to move the figure in different directions when I press different buttons? We should start by limiting this program to only move the figure upwards when I press the ‘w’ key, specifically.”

Next she’ll explain that processing automatically creates a variable for us, called ‘key’ that always equals the name of the last key that was pressed. So we can refer to this variable inside our keyPressed() function to check that the key that was pressed was indeed the ‘w’ key before moving the figure. Now she can complete the keyPressed() function by adding the conditional as defined above, and demonstrating that the figure only moves upwards when she presses ‘w’.

Observations Coded for Cognition

Students will be verbally directed to replicate this code, then to use the same positioning variables to draw a different figure of their own choosing, so they now move their own shape upwards when hitting ‘w’. Next, students will be asked to make their figure move downward when pressing ‘s’. Next, they will be asked to make their figure move left when pressing ‘a’ and right when pressing ‘d’. Each time the students are asked to write code, the instructor will leave the example code displayed on the projector, and will wander about the room, looking at students’ code and asking them about their progress. This questioning is intended to identify students who need help, as well as to gauge what students find difficult, and what they find interesting.

Rubric – Isolated Implementation

When asking for help, student can articulate what they are trying to achieve (intent), how they have attempted it (methods), and what has happened (results) (MK0)	1: Student can only explain one or none of their intent, methods, results.	2: Student accurately describes 2/3 out of their intent, their methods, and their results.	3: Student accurately describes their intent, methods, and results.

Student is unashamed to ask for help (CD0)	1: Student does not ask for help when stuck	2: Student asks for help, but expresses signs of embarrassment in tone, body language, diction, etc.	3: Student will ask for help when struggling, without showing signs of embarrassment.
Student chooses a compound figure to draw. (MK1)	1: Student will not suggest compound figure and must be assigned one.	2: Student suggests a simple abstract compound figure (e.g. circle in a square)	3: Student suggests a complex compound figure, or one that represents something (e.g., house, face, stick figure)
Student is able to create a setup() and draw() functions to prepare their program for complex drawings. (CS5)	1: Student does not produce working setup or draw functions without help	2: Student correctly defines either setup() or draw(), but not both, without help	3: Student correctly defines setup() and draw() functions unassisted
Student initializes global variables for position in setup() function and refers to them globally in the draw() function (CS6)	1: Student does initialize global variables in setup() without help	2: Student initializes global variables in setup() but fails to refer to them globally in draw() without assistance	3: Student correctly initializes global variables in setup() and refers to them in draw()
Student draws a composite figure in a position specified by variable(s) (CS5, CS6)	1: Student fails to draw a complex figure a ball unassisted	2: Student draws composite figure, but fails to position it with variables unassisted	3: Student successfully composite figure with variable positioning without assistance
Student animates their figure by changing the value of the positional variable(s) (CS6)	1: Student requires assistance to animate the ball at all.	2: Student animates ball, but not in the desired way (ball moves incorrectly, or many copies of the ball appear), without assistance	3: Student successfully animates ball in the correct way unassisted
Student implements conditionals to control animation of the figure as described by their variation (CS0)	1: Student does not create any conditional behavior without assistance	2: Student implements some conditional behavior unassisted, but needs assistance to finish implementation.	3: Student implements correct animation for their variation with conditionals unassisted.
Student defines keyPressed() function and moves their figure upwards when the user presses ‘w’ (CS2)	1: Student does not define the keyPressed() function unassisted	2: Student defines the keyPressed function, but does not correctly animate their figure on key press without assistance	3: Student successfully animates their figure upwards when the ‘w’ key is pressed without assistance

Student implements conditionals to move their figure left, right and downwards when the 'a','d', 's' keys are pressed. (CS2, CS0)	1: Student does not implement movement in the other directions without assistance	2: Student implements movement in at least one other direction, but needs assistance to handle movement in at least one direction.	3: Student implements movement in the other 3 directions unassisted.

Integrated Implementation: Handling User Input (Example)

Relevant Goals: CD0, MD0, CD1, CK1, CS1, MK0

Once students have implemented handling student input in an isolated context, they will implement the same pattern in the context of their game. This provides students the opportunity to deepen their understanding by engaging with the new concept amid richer complexity and to the end of improving something real that the student is invested in (**Authenticity, Motivation, Transfer**). Below is an example from Day 4, focused on handling User Input in students' course projects.

Instruction

Because students have just received instruction and practice handling User Input, the instructor will forego a lecture for this activity, and instead move straight to the performance task. The instructor will tell the students that now that they have seen how user input can be used in isolation, and practiced implementing it themselves, that they will now figure out how to integrate user input into their personal projects. Students will come to expect this part of the class, and likely be thinking about how user input could be useful in their games before this point in the class, because of how the integrated implementation consistently follows the Spiral Review Practice and Isolated Implementation each class, according to the schedule (**Norms**).

Observations Coded for Cognition

The instructor will go around the room, and ask each student how the mechanic will be useful in their game. This gives an opportunity to gauge students' recognition of how this tool applies to the context of their game. This is feasible in light of the small size of the class (up to ten students).

The instructor will then ask students to start building their new features that uses user input. The example code from the isolated example will remain projected on the screen to serve as a worked example[8].

Rubric – Integrated Implementation

Student is able to identify how user input can be used to facilitate a necessary feature of their game (CK1)	1: Student needs assistance to see how responding to user input can be used in their program.	2: Student is able to describe one way that responding to user input could facilitate a feature of their program.	3: Student can describe two or more features that could be added to their program by responding to user input.
When asking for help, student can articulate what they are trying to	1: Student can only explain one or none of	2: Student accurately describes 2/3 out of their intent, their	3: Student accurately describes their intent, methods, and results.

[8] ABC's 303

achieve (intent), how they have attempted it (methods), and what has happened (results) (MK0)	their intent, methods, results.	methods, and their results.	
Student is unashamed to ask for help (CD0)	1: Student does not ask for help when stuck	2: Student asks for help, but expresses signs of embarrassment in tone, body language, diction, etc.	3: Student will ask for help when struggling, without showing signs of embarrassment.
Student defines keyPressed() or mousePressed function to respond to user input (CS2)	1: Student does not define the keyPressed() or mousePressed() function unassisted	2: Student defines the keyPressed() or mousePressed() function, but does not correctly animate their figure on key/mouse press without assistance	3: Student successfully animates their figure appropriately in response to user input unassisted.

Design Criteria Considerations for Performance Tasks

Validity: One challenge to validity is that it is difficult to assess whether students will ask for help and how they feel about asking for help if they are able to achieve success unassisted. This means we may be unable to know both that a student has mastered CS1 and that they are willing, able, and unashamed to ask for help when needed. This will be mitigated by repeating the assessment of these goals in every isolated, integrated and Spiral Review Practice assessment. While students may not need help during any particular assessment, it is highly likely that they will struggle at some point, and it is during these times that the instructor will be able to collect data on their willingness and ability to ask for help. This problem will be further mitigated by checking in verbally with students while they work, and by assessing dispositions through surveys at the end of each class and at the beginning and end of the course.

Equity: While this course is designed to encourage acknowledgement of difficulty and failure, it will likely be very embarrassing for some students to ask for help when they see their peers succeed unassisted. This might be mitigated by encouraging students to help each other, as it may be less embarrassing to get help from a peer than from the instructor, and also by the instructor modeling dealing with mistakes and not knowing how to do things authentically and earnestly to create a norm of perseverance and communal helpfulness, rather than skill-shaming (**Norms**, **Belonging**).

Differences in visualization or more broadly in the speed with which students digest and master the material, will be handled by providing the more advanced students the advanced goals pertinent to functions and classes, while keeping them working on the same course-long project. Through this project-based learning, all the students will have similar tasks to do during work time, even if some students are processing more advanced material. Likewise, if some students require additional time to process previous topics, they can work on implementing these features at a slower rate, while still

working on their games at the same time as the rest of the class. It is important that norms of acceptance and humility be established to avoid boastful skill-shaming, and instead foster helpfulness.

Reliability: The reliability of this assessment will be facilitated through the rubrics and the instructor's experience evaluating students in Computer Science. It can be difficult to track student progress foreach individual student during the performance tasks. To make the writing easier, instructors are encouraged to use a streamlined version of the rubrics that leave extra space for in each row. This way, instructors can write the name of each student in whichever column denotes their performance level for each specified item on the rubric, with room for a small note that justifies the decision (this will likely be a summary of each student's idea, or approach). Below is an example of the streamlined version of the above Spiral Review Rubric, with the criteria for each performance level summarized to make additional space for notes. Instructors are expected to grade according to the longform criteria of the rubric as expressed above, but may find it easier to track student achievement in this abbreviated version. Reliability can be further improved if an additional instructor is available to cross-compare scoring of student performance on the three performance tasks. Matching scores from multiple instructors is generally a good indicator of assessment reliability.

Student is able to come up with a novel variation on the bouncing ball (MK1)	1: Must be told	2: With Scaffolding	3: Independent
Student is able to come up with their own composite figure to draw (MK1)	1: Must be told	2: With Scaffolding	3: Independent
When asking for help, student can articulate what they are trying to achieve (intent), how they have attempted it (methods), and what has happened (results) (MK0)	1: None/One	2: 2/3	3: 3/3
Student is unashamed to ask for help (CD0)	1: Needs help to start	2: Bashful	3: Unashamed
Student is able to create a setup() and draw() functions to prepare their program	1: Needs help to start	2: 1/2	3: Both Indepenent

for complex drawings. (CS5)			
Student initializes global variables for position in setup() function and refers to them globally in the draw() function (CS6)	1: Needs help to start	2: setup() not draw()	3: setup() and draw()
Student draws a composite figure in a position specified by variable(s) (CS5, CS6)	1: Help with both	2: Draws can't position	3: Draws and positions
Student animates their figure by changing the value of the positional variable(s) (CS6)	1: Needs help to start	2: Animates incorrectly	3: Independent
Student implements conditionals to control animation of the figure as described by their variation (CS0)	1: Needs help to start	2: Incorrect use	3: Independent

Informal Evidence

The questions student ask during lab will be useful for assessing what they do not understand, as well as their willingness to ask questions, their topics of interest, and their ability to imagine what could be built. These conversations should be interleaved during student practice activities and serve the dual roles of assessment and providing feedback[9] to students about their dispositional growth. This will allow instructors to establish the dispositional goals of the course as norms[10] which are continually checked in on by both students and instructors so students can be encouraged to grow.

Observations Coded for Cognition

Questions the instructor is encouraged to ask include:

1. What are you trying to do? What have you tried? Where are you stuck? (MK1)
2. What's good about your program? (MS0)
3. What would you like to add? (MS1)
4. How could your program be better, even if it did the same stuff? (MS2)
5. Is there anything you'd like to ask me about? (CD2)
6. How is your grit/inquisitiveness/interest today? (MD1)
7. Though of anything cool you'd like to make some day? (CD1)
8. Tell me about [this tool]. What is it for? How do you use it? (CK0-CK5)

Interpretation/Rubric

Student identifies what was difficult for them and provides an explanation for why this step was difficult (MK0)	1: Student is unsuccessful in reporting any difficulties.	2: Student is able to report what was difficulty, but not why.	3: Student reports what was difficult and explains why it was difficult.
Student accurately identifies a well-implemented feature in his or her program (MS0)	1: Student fails to identify a well-implemented feature in his or her program.	2: Student identifies a well-implemented feature in his or her program without explaining what is good about the implementation.	3: Student identifies a well-implemented feature and explains something good about it (e.g clarity, efficiency, novelty, etc.)
Student identifies a desirable feature to add to his or her program (MS1)	1: Student fails to identify a desirable feature worth adding to his or her program.	2: Student identifies a desirable feature but cannot explain how it could be built.	3: Student identifies a desirable feature and describes on a high level how it could be implemented by providing which data and control structures (variable types, and conditionals/loops) would be needed.

[9] ABC's 76
[10] ABC's 178

Student identifies a means of refactoring his or her code for clarity, modularity, or efficiency (MS2)	1: Student fails to identify a place where his or her code could use refactoring.	2: Student identifies a place where code could be improved, but cannot describe what should be done to implement the refactorization.	3: Student identifies a place where his or her code could be refactored and describes how this could be done.
Student expresses curiosity about Computer Science. (CD2)	1: Student has no questions.	2: Student asks a single question without follow up.	3: Student asks a question, and then a related question informed by the answer to the first question.
Student accurately assesses his or her own grit, interest, or inquisitiveness	1: Student will not say or claims not to know about their disposition(s)	2: Student gives an answer that appears unreflective of their actual attitudes to the instructor.	3: Student gives an answer that appears to accurately reflect their disposition that day.
Student accurately describes role and means of implementing inquired tool (CK0-CK5)	1: Student cannot describe reason or method for implementing the tool	2: Student can accurately describe either the reason or the method for implementing the tool	3: Student can accurately describe both the reason and method for implementing the concept.

Design Criteria Considerations

Validity: One challenge to validity is that the questions written above are informal, and so students may wander off in conversation without directly answering them, or otherwise answer them in a way that does not map neatly onto the above rubric. It is therefore more likely that a student be able to express mastery of the goals listed above but fail to perform at their ability than it is that students would answer convincingly without possessing the desired understanding (as the rubric should enable inference that a student possesses the desired understanding in the case that they successfully fulfill a criterion). This challenge can be mitigated by the instructor steering the conversation towards questions that are useful for assessment. Additionally, the informal nature of this assessment and its placement during all three performance tasks will invite many opportunities to ask students questions, and to build up an understanding of their levels over time.

Another challenge for validity is that there may not be opportunities to speak with every student in adequate depth about each question every day, due to time constraints. This is mitigated firstly by keeping the class size small (<=10), and secondly by building up an assessment of students' standings on each assessment item over the course of the week. While not every student may get fully assessed informally every day, every student should be assessed according to this full rubric by the end of the course.

Self-Assessment

End of each Class Survey

Observations Coded for Cognition

At the end of each class, students will answer a short survey, consisting of the following questions:

1. What was difficult about today's lesson? Why? (MK0)
2. What was the coolest thing you made today? (MD0)
3. What feature would you like to add to your game? (MS2)
4. How could your code be better, even if it looked the same while it was running? (MS2)
5. What is user input, and could you use it to do? (CK1 – This will shift to address the day's lesson)

Interpretation/Rubric

Student identifies what was difficult for them and provides an explanation for why this step was difficult (MK0)	1: Student is unsuccessful in reporting any difficulties.	2: Student is able to report what was difficulty, but not why.	3: Student reports what was difficult and explains why it was difficult.
Student reports a feature or component that they created that they are proud of. (MD0)	1: Student does not report anything they are pleased with	2: Student names something they are proud to have built without providing detail about what it was or why it is cool.	3: Student describes something they are proud to have built and explains why it is cool/why they are proud to have built it.
Student identifies a feature that they would like to add to their game (MS2)	1: Student does not identify a feature they would like to add to their game.	2: Student identifies a feature they would like to add to their game by naming it without providing explanatory details	3: Student identifies a feature they would like to add to their game with supporting details
Student identifies ways to refactor their code by making it clearer (comments/reorganization), making it more efficient (performing fewer computations), or replacing redundant code (MS2)	1: Student's refactoring suggestions would change the behavior of their game, or would not improve the organization of the code, or student does not suggest a means of refactoring.	2: Student suggests one way to refactor their code for improved clarity, efficiency, or reduced redundancy	3: Student suggests two or more ways to refactor their code for improved clarity, efficiency, or reduced redundancy
Student explains that user input is when the person using a computer interacts with the computer by clicking the mouse, or typing on the keyboard, and that responding to user	1: Student incorrectly explains what user input is, or does not attempt to explain what user input is	2: Student explains what user input is, without providing a use-case	3: Student explains what user input is, and provides a use case for handling user input

input lets you make your program do different things when the user clicks or types. (CK1)			

Design Criteria Considerations

Validity: This survey shares the same potential pitfalls and mitigating factors to validity as the end-of-course survey

Reliability: The reliability of end-of-each-class surveys will be facilitated through the use of the above rubric, which seeks to render the assessment criteria objective and measurable.

Beginning and End of Course Survey

Observation Coded for Cognition

At the beginning and end of the course, students will answer a dispositional survey:

1. What computer program(s) would you like to build? (CD1)
2. What would you like to know about computer programming? (CD2)
3. What makes a program good? Why? (MD0)
4. What is failure? Is it bad? (CD0)

Interpretation/Rubric

Student describes one or more computer programs he or she would like to build (CD1)	1: Student's answer appears insincere, or fails to describe a computer program they would like to build.	2: Student describes a computer program they would like to build, but the description is vague, poorly defined, or impossible	3: Student sincerely describes a computer program that they would like to build with sufficient clarity that the instructor can explain the function of the envisioned program to another person.
Student expresses sincere interest in learning more about a topic in computer science. (CD2)	1: Student's answer is insincere or fails to describe a topic in computer science that they would like to learn about.	2: Student describes a topic they would like to learn about, but in a way too vague for the instructor to indicate how the student might procure resources to learn more about the desired topic.	3: Student describes a topic they would sincerely like to learn more about with sufficient clarity for the instructor to indicate to the student how the student might go about learning about their topic.
Student identifies features that make a program good, such as usefulness, engagement,	1: Student claims not to know what makes a program good, or produces an	2: Student describes at least one quality that makes a program good, but does not	3: Student describes at least one quality that makes a program good

efficiency, clarity of purpose and function (MD0)	incomprehensible, or unreasonable answer (e.g. a program is good if it is blue)	describe why this quality is desirable.	and explains why this quality is desirable.
Student exhibits positive outlook on failure and explains how failure can be useful. (CD0)	1: Student fails to define failure intelligibly, or reports that failure is bad/a reflection of incompetence	2: Student defines failure and exhibits a positive attitude towards failure, but does not explain what could be useful about failing.	3: Student meaningfully defines and exhibits a positive attitude towards failure, and explains how failure can be used to grow personally, or to improve a creation.

Design Criteria Considerations

Validity: Self-reported assessment always entails the possibility that students will misrepresent themselves, either out of a lack of metacognitive awareness, or a desire to game the system and provide answers they expect that the instructor wants to hear. This will be somewhat mitigated by the informal context of the course. As grades are not provided, students have little incentive to misrepresent themselves, as they have no risk of punishment for producing "wrong answers."

Reliability: The reliability of this assessment will be facilitated by the use of the above rubric and the instructor's experience evaluating students in computer science.

Equity: Different students will enter with differing visualization and abstraction skills, which may impact their ability to imagine potential applications of computer science and circumscribe their ability to cultivate and describe their interest in the discipline. By leaving these assessment questions agnostic to any particular topics in computer science (it doesn't matter what the student expresses interest in, so long as it is within the domain), this assessment facilitates fair assessment of students with varying prior knowledge and ability. Students may also enter with varying levels of grit and perseverance. Assessing this disposition in the final question of the entrance and exit surveys will enable the instructor to track whether students develop growth mindsets during the course, which can be used to inform course re-design and iteration.

Part E: Evaluation Research Design

Active Ingredients

The sequence of three performance tasks, the Spiral Review Practice, the Isolated Implementation, and the Integrated Implementation are expected to be the most differentiating and impactful aspects of this course. Together they ensure that each tool is thoroughly practiced and reviewed in varied contexts which give them meaning and depth, while fostering student interest and creativity by empowering them to Participate in Making.

Research to Evaluate Educational Implementation

Instructor Implementation

The course implementation can be evaluated in part by checking the thoroughness with which assessments were documented for each student. Has every student been assessed on the Pre and Post Course Surveys? Has every student been evaluated on each of the 5 End of Class Surveys? Has every student been evaluated with Spiral Review Practice, Isolated Implementation, and Integrated Implementation all 5 times? Has informal evidence of goal progress been documented for each student throughout the course on the conversational rubric? For each of these assessments, has the instructor provided evidence of how the student met the criteria that warranted their grade that was given, (e.g has the instructor left a note of what the student has done?) Can a second instructor who is provided with the results of the assessments for these students independently verify them against the original instructor's ratings, and do they generally agree? All of these questions can be used to gauge the extent to which the course has been faithfully implemented. Below is an example of one page of the abbreviated rubric for the Spiral Review activity, which could be evaluated by a separate instructor to determine the fidelity with which the course instructor implemented the course.

Student is able to come up with a novel variation on the bouncing ball (MK1)	1: Must be told Sam- tried but failed Sally- didn't engage	2: With Scaffolding JO- told to think about direction Jim- wanted to be random, I gave suggestion to teleport at edge Jay- wanted to "go fast" Geoff- "make it done"	3: Independent Sarah - come through other side Cris - bounce at 45° Sasha - go around border Amy - Bounce up & down, side to side
Student is able to come up with their own composite figure to draw (MK1)	1: Must be told JO - no idea Jim - no idea Jay - no idea	2: With Scaffolding Sam - "Some comic character" Geoff - "A guy" Sarah - something w/ triangle Amy - two circles	3: Independent Sasha - the triforce Cris - Spiderman Sally - a bow
When asking for help, student can articulate what they are trying to achieve (intent), how they have attempted it (methods), and what has happened (results) (MK0)	1: None/One Jay - Struggled w/o asking couldn't point to problem	2: 2/3 Sally - I want it to go to the other side, but it sticks to the wall Geoff - want my guy's arms to go up, but they go down. Jim - I tried this routine) call here, but it never gets Sarah - why doesn't it appear on the other side?	3: 3/3 Amy - I want it to move up and down, but it only goes sideways Here's where I change my variables Cris - I can't get the angle right, it's too wide. Here's my calculation
Student is unashamed to ask for help (CD0)	1: Needs help to start Jim - Struggled w/o asking	2: Bashful Sally asked shyly Geoff asked shyly after a while	3: Unashamed Amy spoke up Cris asked politely Sarah asked Jim asked
Student is able to create a setup() and draw() functions to prepare their program	1: Needs help to start	2: 1/2 JO - forgot colon Sarah - indent error Jay - forgot setup	3: Both Indepenent Sasha Jim got it Cris Geoff Amy Sam

Student Implementation

Student seriousness can be gauged informally by the instructor, based on verbal interactions, and the focus and engagement the students demonstrate in discussion and during the programming activities. During the research study, this could be evaluated by reviewing video data by the researcher responsible for evaluating the implementation of the study itself. Student engagement may also be indicated by survey results targeting CD1, as measurements of student interest will likely correlate with the depth of their engagement and sincerity of their efforts.

Research to Evaluate Educational Impact

Research Questions

This study explores two research questions:

1. What effect does increased Participation and Making have on fostering interest compared to Observation and Worked Examples?
2. What effect does increased Participation and Making have on knowledge and skill learning Observation and Worked Examples?

Experimental Design

The experiment will vary instructional activities during the Spiral Review and the Isolated Implementation to measure the impact on the surveys and Integrated Implementation.

During the Spiral Review, subjects in the control group will receive two worked examples demonstrating when and how to implement all the tools learned heretofore. In contrast, subjects in the treatment group will receive one worked example and will then practice implementing their own variation, as described above in the Assessments and Instruction section. Likewise, during the Isolated Implementation, the control group will receive two worked examples and the treatment group will receive one worked example and then practice by creating their own variation on the worked example, as described in the Assessments and Instruction section. The independent variable is thus the use of increased worked examples vs the combination of one worked example with practice.

The dependent variables will be student scores on the surveys (both daily and pre and post-course), as well as the Integrated Implementation. These activities are to be presented identically to subjects in both groups.

Methods

In order to avoid tension between students assigned to the control vs treatment group, each class in its entirety will be assigned to either treatment or control. That is to say that all students in the same class will perform the same activities and see the same instruction, to prevent students from feeling left out by knowing that other students are receiving different instruction and practice. The same instructor will instruct all classes in both groups.

The instructor is to give both classes instruction that is as identical as possible, except for the use of two worked examples vs one worked example + practice in the Spiral Review and Isolated Implementation activities. The control group should spend as much time on the two worked examples in each activity as the treatment group spends on the worked example + practice.

The experiment will take place throughout the week-long duration of the course, with data collection scoring, and analysis performed after completion of the course.

Data Collection & Scoring

In order to mitigate instructor bias in scoring student work, scoring will be performed independently by a 2nd and a 3rd researcher who are ignorant to which class is treatment vs control. Surveys will have names replaced with ID's unknown to the evaluating researchers, but which can later be re-correlated to the student who took the survey. Pre and post course surveys will be marked with an anonymizing ID so the evaluating researchers do not know which surveys were taken at the beginning of the course, and which were taken afterwards. During the week-long course, the instructor will video and audio record the entire class, in both treatment and control groups, and send the evaluating researchers the recordings of only the Integrated Implementation activity (to avoid revealing which class received which instruction). The evaluating researchers will then grade the student's performance on this activity independently of the instructor and each other, based on the video data and the code produced by students, and without knowing which class is treatment vs control.

A fourth researcher who has access to the ID's correlating the students to their survey data and who can disambiguate pre vs post course surveys will then correlate the survey scores to each student. For each item assessed on the rubrics for the surveys and the Integrated Implementation, this fourth researcher will tabulate each student's scores at every opportunity (averaged across both evaluating researchers), and analyze the progression of the students' scores. Student growth on each assessment item at each opportunity will then be assessed as the score on that opportunity – the score on the students' first opportunity. Average gain at each opportunity will then be analyzed and compared between treatment and control conditions.

Hypotheses and Related Predictions

I hypothesize that subjects in the treatment group will show significantly increased gains on skill and dispositional assessment items on average compared to subjects in the control group, due to increased motivation and sense-making from the processes of participation and making (**Participation, Making**, **Motivation**).

Assessment of Design Quality

Sampling will not be entirely random, because every student in a given class will be assigned to the same group as every other student in that class. This has the ethical advantage of discouraging student resentment of differential treatment, by keeping subjects ignorant of the instructional differences applied to treatment and control groups, but the disadvantage of making the sampling less random and introducing potential instructional differences besides the desired difference of two worked examples vs one worked example + practice (such as spending more time during class explaining one concept vs another). This can be mitigated in several ways. First, this can be addressed via sample size. The more classes that are included in the studies, the less likely it is that random variances in instructional differences between treatment and control groups will influence the measured data. Second, having the same instructor teach all classes will help to control for unintended differences in instruction. Third, the full video data, including the instruction withheld from the evaluating researcher, will be reviewed by a fifth researcher who can thus evaluate the equity of instruction given to the treatment and control groups.

Time spent on instruction will be controlled by the instructor by keeping the Spiral Review (including worked examples and/or practice) limited to 30 minutes, the conceptual overview of the new topic limited to 15 minutes, and the Isolated Implementation (worked examples and/or practice) limited to 30

minutes, as allocated above. This can be verified by the fifth researcher who is responsible for evaluating the identical nature of instruction for all classes in both groups.

Cross-assessor reliability will be maintained through the use of two separate evaluating researchers who independently score student performance. The fifth researcher who is evaluating the implementation of the study can compare the scores across the two evaluating researchers' and the instructor's scorings for each student to identify signs of poor reliability.

As with the course irrespective of this study, the relative simplicity and objectivity of the three-tiered rubrics (generally corresponding to complete dependence, partial independence, and the ability to independently perform the assessment item without scaffolding), will help to ensure the validity of the inferences made from student scores and gains. Triangulation will be similarly be achieved through the repeated and varied assessment opportunities for each subject.

Reflection

Self-Assessment of the Product

1. My goals, assessment, and instruction are well aligned. The conversation with professor Kosbie was very helpful in focusing my entire project around the fundamental goal of fostering interest. I believe that all of my goals, together, serve this purpose by empowering students to create things that excite them and to imagine and wonder what they are able to create. My assessments target each of my goals from varying angles multiple times across the course. My instruction shows students not only how but why to use the tools taught in the course, with multiple reminders and opportunities to practice across multiple days.
2. My age level helped to point me towards a course duration of 5 days over the summer, and towards the fundamental purpose of fostering interest, around which the entire course is centered. It also lead me towards the application of game building as a context to practice programming, as this is a domain generally interesting to many students in this population. Older students would have been likely to enter with more interest in computer science as a professional skill, which would have changed the course focus to one of empowerment through knowledge and skill, rather than using knowledge and skill as a means of cultivating interest. Younger students would likely not have been ready for written programming languages, so the context would need to be changed to use a visual platform like SCRATCH.
3. I believe the descriptions included in this report are thorough and rigorous. The greatest challenge in implementing this design would likely be the course pace. The structure and pace for each day may be unreasonably fast for students new to programming, and the large amount of time spent each day, even with a break, may be too much for students this age to endure. This feature is worth evaluation iteration in implementation. Since none of the designs included in this project have been directly tested, the practicality of implementing it is perhaps the largest weakness of the project. I believe it is achievable, but only running the course could determine this with certainty.
4. Instructors implementing this design should already be familiar with programming, and with conceptual knowledge of the programming tools (drawing, variables, conditionals, user input, loops, lists, functions, and classes) involved in the course, as well as the processing library, and how to create python programs in processing. Additionally, instructors should be well enough versed with mac, linux, and windows operating systems to help students install processing and

setup their file structure to store, find, and run programs. Pedagogically, instructors should be familiar with the big ideas referenced in this document, and particularly how to teach using worked examples and to foster learning through making.

5. The most innovative features of my design are the way the programming assessments provide students repeated practice making things with programming in an authentic and engaging context. Students will deepen and reinforce their understandings of core concepts through repeatedly practicing them in new contexts. Additionally, while this has been done before, the course also heavily emphasizes the role of visualization and interactivity via animation in rendering abstract programming concepts concrete and intelligible.
6. Based on Hao's feedback during the poster draft session, I changed my presentation style to give more information about the role of interest in my project and how it is assessed, in addition to updating my poster layout significantly to be easier to read and to demonstrate clearer hierarchy of information.

Self-Assessment of Process

1. I had a relatively clear picture of who I wanted to teach and what my classroom context would be like from early on. This helped my design process stay focused to my learner's needs. My conversation with professor Kosbie was especially helpful in orienting my goal focus, which greatly informed the entire project. I think I came to a relatively stable set of well-specified goals relatively early in the process, and this made the whole design process much smoother than it otherwise could have been. Additionally, I think I responded well to feedback, and sought help when I needed it, much to my benefit. I feel that I did a good job of addressing critiques of my design in depth, rather than in a perfunctory manner. One weakness of my process is that I didn't seek out examples from other designers of how the phases could be done well and properly. I generally worked straight from the guidelines without seeing how other people approached these tasks. I tend to err on the side of doing things my own way to excess, rather than overly imitating others' work, and this may have made my process more cumbersome and less effective.
2. I struggled with how to assess dispositions. I believe the surveys and informal evidence can go a long way to this end, but dispositional assessment is something I was completely unfamiliar with before this project, so it was difficult for me and I'm unsure how well this design could work in that regard. I also had difficulty balancing my intuitive sense that subjects in the research project should not see that other subjects are receiving differential instruction against the need to ensure that between the experimental and control groups *only* the desired instructional variables were varied.
3. Sharon's suggestion to use surveys to assess dispositions was very helpful in overcoming the first problem (though I would feel more confident in the efficacy of my design if I could implement it myself to evaluate its difficulties in practice). Similarly, my conversation with Sharon about how to keep the experimental scoring double blind, and the principle need to identify factors that might undermine the validity of an experiment in order to mitigate them, or at least circumscribe them, helped with the second problem. Part of my difficulty with the research design was I was initially imagining the course to take place on a relatively low budget, and with only one involved instructor, as this mirrored my own experience. Opting to include more researchers with differing responsibilities and differing access to information on the research opened many doors for improving the validity of the study.

4. Peer feedback was helpful to give and receive. The opportunity to think critically about others' work was useful in determining what worked and what didn't in my own design. Additionally, suggestions I received from my peers were always constructive and useful to consider.
5. The lessons learned in the project will be useful in the future for my instructional design. In particular, I will continue to use backwards design, heavily emphasizing the value of strong goals from the outset of each instructional design project, and the value of iteration. I really enjoyed taking each opportunity to add a new section as additionally an opportunity to revise all the work put in heretofore. This was a very useful process for continually realigning the project's goals, assessments, and instruction. I intend to further pursue instructional design in professional contexts, and intend to use this backwards and highly iterative process.
6. Next time, I hope to seek and incorporate more peer feedback earlier in the process, as well as to test small portions of the design on learners in my target population. More outside perspectives will help to ensure the project has been considered from angles that I will not come up with on my own. Testing the design early and often will help to ensure the practicality of this course, and to ensure alignment with actual learners in context.