Summary

Introduction to Computer Science

Welcome to the TEALS Intro CS Curriculum!

The **Introduction to Computer Science curriculum** is a flexible and approachable course adapted from the UC Berkeley CS 10, and is course for a wide range of high school students from diverse backgrounds. The course has been successfully implemented in hundreds of high schools. Introduction to Computer Science is an engaging course that explores a variety of basic computational thinking and programming concepts through a project-based learning environment. Every unit culminates in a comprehensive project and roughly 75% of student time is spent building projects and practicing the skills they are learning.

Visual and approachable

This course uses Snap!, an approachable visual block-based programming language with a robust tool set, perfect for introducing students to coding for the first time.

- GitHub: https://github.com/TEALSK12/introduction-to-computer-science
- GitBook: https://tealsk12.gitbooks.io/introduction-to-computer-science/content/

Introduction to Computer Science Implementation Options

The Introduction to CS course can be offered as a semester-long course offered twice in a single school year or as a year-long course with an expanded curriculum. The year-long class transitions to text-based programming using the beginner-friendly Python language in the second semester. Teachers participating in the TEALS Program can use the following options:

- TEALS Second Semester Introduction to Computer Science - Python curriculum
- Carnegie Mellon CS Academy.

TEALS Curriculum Changelog

Version: [2.1.1] Last updates: 2018-08-21 Changelog.md

About this curriculum

Creative Commons Attribution Non-Commercial Share-alike License

Philosophy

This curriculum has been designed by the TEALS program to support computer science teachers and/or volunteer professionals teaching an introductory computer science course in a high school classroom. The curriculum is based on, and borrows heavily from, the Beauty and Joy of Computing Curriculum developed at the University of California, Berkeley. The TEALS curriculum has a heavier focus on the basic programming components of the course than BJC, sacrificing some of the advanced programming and conceptual topics that are less appropriate in an introductory high school classroom.

This curriculum advocates a "hands-on" learning approach in which students' primary means of learning is through discovery, experimentation, and application. To that end, each unit is built around a large, culminating, programming project that exercises the objectives of the unit. In addition, nearly all lessons in the curriculum include a guided activity of some kind to allow students to practice with and experience the concepts covered in the lesson first-hand. Taken together, the lessons provide the skills and support necessary to enable students to complete the project and demonstrate mastery of the unit's objectives. Substantial class time should be provided for the project in each unit to ensure students have the opportunity to demonstrate mastery of the skills from each unit before moving on.

Because this curriculum was designed to be used in a wide variety of classrooms, we have made as few assumptions as possible. In particular, the curriculum does not depend on any specific technologies or resources in the classroom other than computers with reliable internet access. This curriculum is also designed without any student homework assignment, as not all classes will have students who can reliably access a computer with internet access at home. For classes where home computer access is not an issue, some amount of lab work can be reassigned as homework.

Resources

This curriculum is designed for use with the Snap! programming language designed at the University of California, Berkeley. Snap! is a visual programming language designed to allow students to focus on concepts and skills rather than syntax when learning to program. Snap! is an extension and reimplementation of Scratch, designed at MIT, and many Scratch lessons and programs can be easily adapted to Snap! The following resources are available to support use of Snap! in this curriculum:

- The Snap! Reference Manual
- Snap! examples and extensions (including using hardware devices with Snap!):
- ScratchEd online community for Scratch educators:
- Beauty and Joy of Computing curriculum:

Snap Mirrors

Access to Snap! is necessary for students to complete the labs. The main Snap! site has been known to be inaccessible due to system updates or network outages. It is important to have contingency plans in the event the web site is unreachable. Here is a list of mirror sites that can be used in the event the main site is unreachable. In addition, Snap! can be downloaded to run locally on a student's computer, however the projects will not be able to be save to the cloud and will need to be exported and then imported to the cloud when Snap! becomes available.

The Berkeley Snap! is located at: http://snap.berkeley.edu/

The following are mirror sites:

- Mirror 1
- Mirror 2
- Mirror 3

Download a local copy of Snap! as a backup:

1. Run Snap! from browser
2. Click on the Snap! logo in the upper-left of the app.
3. Choose "Download source" from the menu

   Snap! download
4. Save snap.zip locally on your computer.
5. Extract snap.zip.
6. Open snap.html in a web browser.

Using this curriculum

Semester pacing

The TEALS Intro CS curriculum is designed for a semester-long introductory course meeting daily for 55 minute periods. A course that meets these criteria is expected to have roughly 90 class meetings in a semester. However, the TEALS curriculum in its current form includes an estimated 102 days of material. First and foremost, teachers should be reminded that the pacing in the curriculum is intended as a guideline and teaching teams are encouraged to make any adjustments they deem appropriate or necessary. However, there are certain aspects of the curriculum that are vital to maintain. In particular, the following should be considered when making pacing adjustments:

Lab Days

- The number of lab days allotted for each project is an estimate for a typical class. Classes that are moving quickly, or classes in which most students are able to do work outside of class, can reduce the number of in-class lab days. If this route is taken, be certain that students still have enough time available to complete the projects. As the projects are the primary summative assessments in this course, it is vital that students not be rushed through completion of the project and that requirements are not cut in an attempt to shorten the time necessary.

Culture Days

- Culture Days (see below) are included in the curriculum roughly once every two weeks. While it may be tempting to reduce the number of culture days, or remove them entirely, to gain back class days, these lessons are considered central to the student experience in this course. If an extra day or two are needed, culture days may be skipped on occasion, but teaching teams are advised to avoid making a habit of skipping culture days.

Pacing Considerations

Unit 5 (Optional) It is expected that many teams will find it necessary to remove some or all of **Unit 5** from the curriculum. This unit covers cloning and prototyping, an interesting and worthwhile, though advanced topic. If time allows, teaching teams are encouraged to attempt to include at least part of this unit, possibly with a simplified version of the project, in their curriculum. But this unit can be removed without having too adverse an impact on the student experience, and should be the first major cut if one is necessary.

Unit 6 (Preserved) The capstone experience for the course, Unit 6 enables students to apply the skills they have learned in a large-scale, individualized project setting. Cutting this unit would deprive students of the opportunity to experience a close approximation of a real-world development setting. Earlier units should be condensed or cut as necessary to ensure that Unit 6 is still included in the curriculum.

Daily lesson plans

Most lesson plans in this curriculum are designed to represent a single 55-minute class period with average pacing. Each class will have slightly different needs, possibly including different period length, student capabilities, classroom interruptions, and more. Teachers and volunteers are encouraged to consider the lesson plans as guidance for one possible use of time to present the material, and to feel free to adapt the lesson plans as necessary to fit the needs of the particular class in which the plans are being applied.

With a few exceptions, each daily lesson plan consists of the following components:

1. Welcome/Announcements/Bell work
    - Five minutes are allotted at the beginning of each day for administrative tasks such as taking attendance, giving announcements, returning work, or other necessary actions. During this time, teachers are encouraged to assign "bell work" (sometimes called "do now" activities) for students to work on.
    - These activities aim to engage students with the subject immediately upon entering the room, and should be short, clear, and able to be completed by all students.
    - Specific "do now" activities are not given in the lesson plans, as they should be chosen by the teacher to reinforce or preview the specific topics with which students have or are expected to struggle most.
2. Instruction/Discussion
    - Most lessons begin with a brief period of instruction on the topic of the day. These sections should be kept as brief as possible—the primary means of student learning in most lessons will be the lab activities.
    - The goals of the instruction section of the lesson should be to motivate the concepts being exercised in the lab and to provide a short demonstration to help students find the necessary parts of SNAP the first time.
    - Teaching teams should vary the ways in which the instruction is presented throughout the course, including class discussions, kinesthetic activities, demonstrations, Socratic seminars, occasional lectures, and other approaches.
3. Activity
    - The largest portion of time in each lesson is dedicated to a guided activity that allows students to explore and practice with the day's key topics. Each activity is broken down into several parts, each of which consists of several steps. In general, the steps in a single section build on each other, and each section covers a new topic or new application.
    - It is intended that the labs be well enough structured for students to work through on their own, but teachers should feel free to interject at appropriate points to assess student progress and provide additional guidance as necessary.
    - On occasion, steps 3 and 4 are repeated for multi-part activities.
4. Debrief
    - After each activity has concluded, time is allotted for teachers to review and debrief the activities with students. In general, there is not enough time, nor is there necessarily the need, to go through the lab

step-by-step. Students should be able to at least partially assess their own progress by verifying that their programs function as specified in the lab.

- Rather than presenting solutions to each step of the lab, teachers are encouraged to use the debrief time to focus on particularly tricky or noteworthy parts of the lab or to discuss areas in which students struggled.
- Debrief time can also be used to compare and contrast different possible approaches to some of the problems, emphasizing that, in most cases, there is more than one valid solution.

- Whenever possible, use examples of student work rather than instructor-created solutions during the debrief—this is an excellent chance to showcase students who solve problems in elegant, creative, or canonical ways.

Special lesson plans

The curriculum includes a few special lesson plans that are intended to be applied at multiple times during the semester. These lesson plans are templates for a particular type of lesson, though the specific topics will vary each time the plan is used.

These special plans include:

Lab Day Lesson

- This lesson plan provides structure for a typical "lab day" (a class day in which students are primarily working on a larger-scale project).

- The lab day lesson plan will most often be used during the culminating project of each unit, but can also occasionally be used for larger lab activities that require being split over multiple days.

- The lab day lesson provides a basic structure for reviewing and reinforcing important concepts before providing time for students to work independently.

- The lesson also provides recommendations for how to deal with questions and struggling students, and how to gauge student progress over the course of a project.

Culture Day Lessons

- These lesson plans provide the basic outline for possible types of "culture days" (class days in which cultural, social, societal, or other topics not directly related to programming are covered).

- These days are vital to the experience of this course, as they provide opportunities for students to connect the skills and concepts they are learning to real-world problems, view their society and culture through the lens of technology, and get glimpses of the types of problems and applications that they could encounter in further study of computer science.

- These lessons also provide an outstanding opportunity for students to bring their own personal experiences or knowledge into the classroom.

- Each culture day lesson plan provides a high-level structure for conducting one of several different forms of a culture day.

- Suggested topics are provided, but teaching teams are encouraged to choose topics that are both of interest to their students and that play to the strengths of the instructors.

- These lessons may require more preparation than the typical lesson to adapt the plan to the particular topic chosen.

Homework

This curriculum does not assign homework as part of its typical lessons. Because this curriculum is intended to be used in a wide variety of classrooms, some of which may include students that do not have regular access to an internet-enabled computer at home, all work is done during class time. In some circumstances, assigning some lab activities as homework can enable the teaching team to regain in-class time for additional lessons or activities, but this must be done with care. In particular, if homework is assigned, arrangements must be made so that any students who do not have the ability to complete the homework at home do not fall behind. Further, it should be expected that some students will not complete the assigned homework and teaching teams must have a way to both assess that homework was completed and ensure the material is reinforced briefly in class.

Quizzes

To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Grading

Student work consists of class participation, daily labs, end of unit projects, and final project. Each classroom teacher is responsible for determining the grading breakdown for their classes. TEALS recommends the following as a starting point as grading guidelines for the introductory computer science course.

Percentage	Description
40	Daily Labs
40	Projects and Large-Scale Labs
20	Participation, Notebooks, etc.

Updates and contributions

This curriculum is considered a living document and is intended to be updated regularly both by the TEALS team and with contributions from the TEALS community. In particular, feedback on teaching teams' experiences in conducting the lessons in this curriculum is always welcome. Teaching teams are also encouraged to share ideas for additional activities, lessons, or projects that can be incorporated into the curriculum guide. Please see the Contributing page for more information about how to contribute to the curriculum.

Printing GitBook

The Introduction to Computer Science GitBook can be printed by navigating to https://aka.ms/TEALSIntroPDF. However, the "Download" button does not work. There is workaround depending on the browser:

- click on the document to enable the pdf menu to show and clicking the down arrow or "Save as Copy"
- right click on the .pdf document and select "Save As"

Intro CS Curriculum Map

Unit 0: Beginnings

Lesson	Objectives	Lab
0.1: The First Day	Identify the class they are taking. List the high-level goals of the course. Describe classroom procedures, rules, and norms.	N/A
0.2: Algorithms	Define "algorithm." Construct algorithms for performing simple tasks.	N/A

Lesson	Objectives	Lab
0.3: Programming Languages	Complete levels in the game LightBot 2.0. Complete small programs in SNAP with guidance. Explain why computer programs are written in specialized languages.	N/A
0.4: Snap Self Portrait	Create a simple "program" in SNAP to describe themselves	Getting to Know You

Unit 1: SNAP Basics

Lesson	Objectives	Lab
1.1.1: Welcome to SNAP	Define and identify "blocks," "scripts," "sprites," and "the stage" in SNAP. Write simple SNAP programs. Describe what simple SNAP programs do without executing the code.	Welcome to SNAP!
1.1.2: Welcome to SNAP (Day Two)	Continued lab	Welcome to SNAP!
1.2: Building Blocks	Name the categories of blocks in SNAP and describe what the blocks in each category do. Describe the function of several common SNAP blocks. Be able to use common blocks to build simple SNAP programs.	SNAP Scavenger Hunt
1.3: Drawing Shapes	Construct simple algorithms to draw shapes. Convert algorithms into SNAP programs.	Squares and Triangles and Stars, Oh My!
1.4: Animation	Animate SNAP sprites using costume changes and movement. Trigger action in other sprites using broadcasts.	Sprites in Action
1.5: Nursery Rhyme Project	Apply basic programming and SNAP skills to create an animated movie, play, nursery rhyme, or other scene. Practice good debugging skills to correct issues as they arise while programming	Project 1: Animated Nursery Rhyme
1.6: Project Day 1	Project Work	Project 1: Animated Nursery Rhyme
1.7: Project Day 2	Project Work	Project 1: Animated Nursery Rhyme
1.8: Project Day 3	Project Work	Project 1: Animated Nursery Rhyme
1.9: Project Day 4	Project Work	Project 1: Animated Nursery Rhyme
1.10: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 2: Loop-de-Loop

Lesson	Objectives	Lab
2.1: Loops	Define “loop” in a programming context. Explain why loops are useful. Implement simple repeat and forever loops in SNAP. Utilize loops to reduce redundancy in code.	Squares and Triangles Redux
2.2: Nested Loops	Use nested loops to solve programming problems.	Another Brick in the Wall
2.3: Inputs and Conditionals	Ask for and receive user input in a SNAP program. Use simple conditional (if and if-else) blocks to alter control flow in a SNAP program.	What Shape Is That?
2.4: Variables	Use variables to track values throughout a program.	Guessing Game
2.5: Boole in the House	Define and identify Boolean expressions and operators. Evaluate Boolean expressions. Utilize Boolean operators (and/or/not) to create compound conditions.	Triangle of All Kinds
2.6: Combining Loops and Conditionals	Combine loops with conditionals to create models with repeated but conditional behavior.	What Goes Up…
2.7: Pong Project	Implement a well-written version of Pong. Practice good style and conventions to create readable and maintainable code.	Project 2: Pong
2.7: Project Day 1	Project Work	Project 2: Pong
2.8: Project Day 2	Project Work	Project 2: Pong
2.9: Project Day 3	Project Work	Project 2: Pong
2.10: Project Day 4	Project Work	Project 2: Pong
2.11: Project Day 5	Project Work	Project 2: Pong
2.12: Project Day 6	Project Work	Project 2: Pong
2.13: Project Day 7	Project Work	Project 2: Pong
2.14: Project Day 8	Project Work	Project 2: Pong
2.15: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)
2.16: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 3: Variables and Customization

Lesson	Objectives	Lab
3.1: Abstraction and Friends	Define abstraction, detail removal, generalization, and procedural decomposition in a computer science context. Describe how utilizing procedural decomposition can improve the readability and maintainability of algorithms and/or code. Recognize opportunities to improve algorithms by abstracting or generalizing parts into sub procedures.	N/A
3.2: Procedures	Build custom command blocks in Snap. Utilize detail removal and generalization to construct blocks that practice abstraction.	Drawing Shapes (Again)
3.3: Customizing I: Arguments	Build custom SNAP blocks that take arguments.	Let Me Check My Calendar
3.4: Customizing II: Reporters and Predicates	Build custom reporter and predicate blocks in SNAP.	If My Calculations Are Correct…
3.5: Platform Game Project	Use loops, variables, and Boolean expressions to implement a Super Mario Bros. style platform game. Practice good debugging skills to correct issues as they arise while programming.	Project 3: Platform Game

Lesson	Objectives	Lab
3.6: Project Day 1	Project Work	Project 3: Platform Game
3.7: Project Day 2	Project Work	Project 3: Platform Game
3.8: Project Day 3	Project Work	Project 3: Platform Game
3.9: Project Day 4	Project Work	Project 3: Platform Game
3.10: Project Day 5	Project Work	Project 3: Platform Game
3.11: Project Day 6	Project Work	Project 3: Platform Game
3.12: Project Day 7	Project Work	Project 3: Platform Game
3.13: Project Day 8	Project Work	Project 3: Platform Game
3.14: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)
3.15: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 4: Lists

Lesson	Objectives	Lab
4.1: Intro to Lists	Explain the concept of a “list” in a programming context. Identify scenarios in which lists are useful	[none]
4.2: Static Lists	Create static lists in SNAP. Access elements of a list. Add and remove elements from a list.	You Talkin’ to Me?
4.3: List Practice I	Traverse a list, accessing each element one at a time. Perform operations combining all elements in a list. Select defined subsets of elements in a list.	Guess Who
4.4: List Practice II	Traverse a list, accessing each element one at a time. Perform operations combining all elements in a list. Select defined subsets of elements in a list.	Number Cruncher
4.5: Sequential Search	Explain the sequential search algorithm. Implement several variations of sequential search.	It’s Around Here Somewhere
4.6: Word Guessing Game Project	Use lists to implement a complete version of a word guessing game. Exercise good programming practices to produce code that is not only functional but also elegant and well-written.	Project 4: Hangman
4.7: Project Day 1	Project Work	Project 4: Word Guessing Game
4.8: Project Day 2	Project Work	Project 4: Word Guessing Game
4.9: Project Day 3	Project Work	Project 4: Word Guessing Game
4.10: Project Day 4	Project Work	Project 4: Word Guessing Game

Lesson	Objectives	Lab
4.11: Project Day 5	Project Work	Project 4: Word Guessing Game
4.12: Project Day 6	Project Work	Project 4: Word Guessing Game
4.13: Project Day 7	Project Work	Project 4: Word Guessing Game
4.14: Project Day 8	Project Work	Project 4: Word Guessing Game
4.15: Project Day 9	Project Work	Project 4: Word Guessing Game
4.16: Project Day 10	Project Work	Project 4: Word Guessing Game
4.17: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)
4.18: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 5: Cloning

Lesson	Objectives	Lab
5.1: Intro to Cloning	Explain why prototyping and clones can be useful. Describe how complex goals can be accomplished using cloning.	Connect the Dots
5.2: Cloning Sprites	Demonstrate the difference between sprite and global variables. Explain how cloning and prototyping simplify working with numerous similar sprites in the same program. Create prototype sprites and clones of the prototype sprite. Explain the difference between a “master” sprite and a “clone” sprite.	Lots of Balls
5.3: Communicating With Clones	Pass information to individual clones. [Optional] Describe a race condition that might occur due using global variables and clones. Delete clones when they are no longer needed.	Fewer Balls
5.4: Space Invaders	Use cloning to implement a complete version of “Space Invaders.” Exercise good programming practices to produce code that is not only functional but also elegant and well-written.	Project 5: Space Invaders
5.5: Project Day 1	Project Work	Project 5: Space Invaders
5.6: Project Day 2	Project Work	Project 5: Space Invaders
5.7: Project Day 3	Project Work	Project 5: Space Invaders
5.8: Project Day 4	Project Work	Project 5: Space Invaders
5.9: Project Day 5	Project Work	Project 5: Space Invaders

Lesson	Objectives	Lab
5.10: Project Day 6	Project Work	Project 5: Space Invaders
5.11: Project Day 7	Project Work	Project 5: Space Invaders
5.12: Project Day 8	Project Work	Project 5: Space Invaders
5.13: Project Day 9	Project Work	Project 5: Space Invaders
5.14: Project Day 10	Project Work	Project 5: Space Invaders
5.15: Project Day 11	Project Work	Project 5: Space Invaders
5.16: Project Day 12	Project Work	Project 5: Space Invaders
5.17: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)
5.18: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 6: Final Project

Lesson	Objectives	Lab
6.1: Design Basics	Identify the key considerations when designing a piece of software. Describe methods for prioritizing features, use cases, and/or scenarios. Explain why design and planning are necessary steps in the software engineering process	Final Project
6.2: Brainstorming and Evaluating	Identify factors to use when choosing between project ideas. Rank a group of proposed project ideas using the identified factors	Final Project
6.3: Spec Writing	Identify the main components of a functional project specification and explain the purpose of each section. Develop a project idea into a full, detailed specification	Final Project
6.4: Building a Plan	Identify the main components of a functional project specification and explain the purpose of each section. Develop a project idea into a full, detailed specification.	Final Project
6.5: Project Implementation	Use the skills developed throughout the course to implement a medium- to large-scale software project. Realistically evaluate progress during software development and identify when cuts are necessary. Prioritize features and scenarios and choose which should be eliminated or modified if/when resources and/or time become limited.	Final Project
6.6: Project Day 1	Project Work	Final Project
6.7: Project Day 2	Project Work	Final Project
6.8: Project Day 3	Project Work	Final Project
6.9: Project Day 4	Project Work	Final Project
6.10: Project Day 5	Project Work	Final Project
6.11: Project Day 6	Project Work	Final Project
6.12: Project Day 7	Project Work	Final Project
6.13: Project Day 8	Project Work	Final Project
6.14: Project Day 9	Project Work	Final Project

Lesson	Objectives	Lab
6.15: Project Day 10	Project Work	Final Project
6.16: Culture Day	Connect CS Unit topics with current events	(see Culture Day LPs)

Unit 0

Lesson 0.1: The First Day of School

Learning Objectives

Students will be able to…

- Identify the class they are taking
- List the high-level goals of the course
- Describe classroom procedures, rules, and norms

Materials/Preparation

- ☐ Class list
- ☐ [Optional] Seating chart
- ☐ Bell schedule and classroom location
- ☐ Unit 0 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance
10 minutes	Course staff introductions
10 minutes	Icebreakers
10 minutes	Course outline
15 minutes	Course norms, procedures, and administrative tasks

Instructor's Notes

- Utilize the plan developed with your teaching team for the first day of school.
- The pacing guide above is a very broad suggestion. You should adapt (or ignore) as necessary to fit your team's plan.
- Suggested topics to cover: . * Names and backgrounds of TEALS team members . * Brief background on TEALS program (i.e. "Why are there so many adults here?") . * Classroom rules, behavior expectations, grading guidelines, late work policies, etc. . * Academic Integrity/Cheating policy . * Introductions/Icebreakers with students
  - High-level learning objectives for course

BJC Lecture Suggestions

(This could be used for a Social Media Safety/Awareness lesson)

BJC Lecture 11:Social Implications II Dr. Gerald Friedland

- Dr. Gerald Friedland Sr. Research Scientist at International Computer Science Institute (ICSI) on Sharing Multimedia and the Impact on Online Privacy) 0:00-1:45
- Introduction to Social Media: The Price of Social Media Use-Stephen Colbert 1:50-6:25
- Observations on Sharing Data and Ineffective Privacy Protection 6:30-7:50

- Social Cause: Collection of Data Across Sites 7:50-10:30
- Multimedia in Internet is Growing 10:35-12:05
- CS Problem: Higher Demand for Retrieval and Organization of Data 12:07-13:05
- Manual Tagging & Geo Tagging 13:05-17:30
- Issue of Tracking & Dangers of Oversharing 17:30-18:31
- Berkeley Multimedia Location Estomation Project 18:31-28:14
- ICSI's Evaluation Results 19:49
- YouTube Cybercasing 20:47
- Privacy Implication of Internet and Data 22: 30
- Person Linking Using Internet Videos 25:45-26:45
- Solutions for Privacy that Don't Work: Think Before You Post! 26:45-28:14

Background Information for Instructors

BJC Video Suggestion: BJC Lecture 1: Abstraction

- Basic concepts of the course: 0:00-7:00
- Introduction of Piazza: 7:00-8:25
- Abstraction: 11:40-15:40
- Generalization: 15:50-20:00
- Summary: 20:05-25:10

BJC Video Suggestion: BJC Lecture 2: 3D Graphics

- SOPA & PIPA: 0:00-1:00
- 3D Computer Graphics Explanation: 1:00-5:24
- 3D Graphics steps outlined: 5:25-5:50
- Modelling (Useful for Lab 2.5): 5:50-11:40
- Animation (Uncanny Valley Explanation): 11:40-16:55
- Procedural Based Motion (Lab 1.1): 16:56-20:00
- Genetic Algorithms: 20:05-25:25
- Lighting and Shading: 25:25-27:10
- Rendering: 27:10- 30:55
- Global Illumination: 30:55-34:21

Accommodation/Differentiation

- In some school's the first day of schools is an altered or shortened schedule.

- Find this out ahead of time and plan for however much time you have available.
- If necessary, split these topics across multiple days.

Forum discussion

TEALS Unit 0 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 16, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Algorithm	A complete, well-defined sequence of steps for completing a task or solving a problem.

Word	Definition
Computer	An electronic machine that can solve different problems, process data, store & retrieve data and perform calculations.
Computer Science	The study of the principles and use of computers.
Computer Program	A sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem.
Debug	A process of locating and removing computer program bugs, fixing errors or abnormalities.
Programming Language	A vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

Lesson 0.2: Algorithms

Learning Objectives

Students will be able to…

- Define **algorithm**
- Construct algorithms for performing simple tasks

Materials/Preparation

Needed for this lesson:

- ☐ Supplies to create a peanut butter & jelly sandwich (ingredients, utensils, plates, napkins, etc.)
- ☐ Paper/writing implements for each group of students
- ☐ Large poster paper and markers will allow for display of the algorithms, but standard paper will work fine
- ☐ Unit 0 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introductory discussion; present activity
10 minutes	Students write first algorithms
5 minutes	Sample algorithm execution
10 minutes	Students debug/rewrite algorithms
5 minutes	Second sample algorithm execution
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Introductory discussion

- Invite discussion about what constitutes a computer, what computers do, and what computer science is.
- An excellent protocol for these types of discussions is "Chalk Talk" or "World Cafe"
- Develop definitions for important terms ("computer," "computer science," "algorithm," "program," "programming language")
- Display each term on the board or projector and ask students to provide key ideas or concepts they know that relate to the term. From this, you can develop a classroom definition. Feel free to have a pre-written definition and guide students to that definition using leading questions.
- You can introduce the idea that the first computers were human (a person who makes calculations, especially with a calculating machine) with the story about Katherine Johnson whose calculations were used for manned and unmanned orbital missions (read more here).

2. Activity

1. Write algorithms

   - In pairs or small groups, students will attempt to develop an algorithm for preparing a peanut butter and jelly sandwich. Specify to students that their algorithm must be complete and detailed enough for a "computer" (the teacher) to unambiguously follow the steps and achieve the desired result. OR ask students to develop a 10 step algorithm to teach a computer to brush their teeth.
   - Give little guidance at this stage, as the confusion (and the errors that are likely to result) will both reinforce the importance of specificity and detail as well as provide entertainment value later on.
   - Algorithms should be written on paper to be shared and reviewed. Students should *not* be writing any code, nor should they develop algorithms "in their heads."

2. Algorithm execution

   - After groups have finished, choose a group and have them read their instructions. Act as a computer and follow each step as literally as possible. If there is ambiguity, or if a step is not possible to complete, point out the error.
   - When an instruction is ambiguous or impossible, interpret the algorithm in the most atypical (and hilarious) way possible. This will reinforce to students that many seemingly clear instructions can be taken many ways.
   - Common errors will include:
     - Failing to open a container before using what is inside
     - **Response:** Try (and fail) to access the inside in a humorous fashion (e.g. try to reach through the bag or jar, acting confused as to why you cannot reach the ingredient inside)
     - Failing to specify in which orientation or position to use something (e.g. "grab the knife" but not by the handle, "put down the bread" but not on the plate)
     - **Response:** use or place the ingredient in an obviously (and humorously) incorrect way (e.g. grab the knife (carefully) by the sharp end, put the slice of bread on the table next to plate, spread peanut butter around the crust instead of on the face)
     - Using instructions that are too broad (e.g. "pick up the bread" to mean a single slice, "put the peanut butter on the bread" to mean spreading a small amount)
     - **Response:** Ask for more detail, or interpret the instruction literally
     - Combining multiple steps into one instruction (e.g. "spread peanut butter on the bread" without specifically opening the jar, putting peanut butter on the knife, using the knife to spread, etc.)
     - **Response:** Ask for more detail
   - Most algorithms will fail. If there is time, repeat the process with one or two other groups.
   - Here is an example video of this activity.

3. Debugging algorithms

   - Spend a brief moment explaining that programming is an iterative process, and that errors are expected. Introduce the concept of "debugging." Then, have students "debug" their algorithms and attempt to fix all errors and ambiguities.
   - Changes should be made on paper. Consider introducing a standard notation for edits (strike-through for deletion, carat (^) for insertion, circle for modify, etc.) to make changes easy to spot.
   - If possible, have students make changes using a different color marker or pen.
   - While students are working, circulate and look for a promising algorithm. The goal is to have a successful algorithm at the end of the class.

4. Execute debugged algorithm

   - Once groups have finished debugging, repeat the execution process. Hopefully, at this point, at least one algorithm will result in a proper sandwich.

   - If no successful algorithm is found, try to debug on the fly. When you hear an incorrect or unclear instruction, point out the error and either propose or request a fix before proceeding. The goal is to create a sandwich before the end of class.
   - Many algorithms will still have similar problems to the first iteration. Others will have too much detail (see below) or other, subtler problems (such as skipping trivial steps like putting the two slices of bread together). Try to take note of issues while circulating so you can address them quickly.

3. Debrief

- Ask students to describe why there were problems with the first round of algorithms, and how those problems were fixed. Encourage the use of computer science terminology.
- Keep students from fixating on the specifics of any one error and guide discussion to the general approaches and concepts used to resolve problems.

- Have students discuss what lessons can be learned from this activity and how they can be applied to programming and computer science.

BJC Lecture Suggestions

BJC Video Suggestion: BJC Lecture 6: Algorithm (With Luke Segar

- Algorithm Definition 3:20-4:20
- Early Algorithms 4:25-5:55
- Familiar Algorithms 6:00-7:30

BJC Video Suggestion: BJC Lecture 7: Algorithm Complexity

- Algorithm Analysis: The Basics 6:00-7:40
- Algorithm Analysis: Running Time 7:41-8:25

Background Information for Instructors

BJC Video Suggestion: BJC Lecture 6: Algorithm (With Luke Segar

- Computer Worms 0:00-1:30
- Algorithm Concept Intro: Rubic Cube Competition 1:40-2:40
- Algorithm Definition 3:20-4:20
- Early Algorithms 4:25-5:55
- Familiar Algorithms 6:00-7:30
- Commonly Used Algorithms (Page Rank, etc.) 8:00-10:45
- Choosing an Algorithm Technique 10:50-12:15
- Ways to Tackle Problems (Brute Force, Top Down, Bottom Up) 12:20-15:30
- Algorithms vs Functions and Procedures 15:30-16:00
- Turing Completeness (Computer Theory-BYOB is Turing Complete) 16:05-21:15
- Algorithm Correctness 21:25-26:00
- Algorithm Summary 26:00-end

BJC Video Suggestion: BJC Lecture 7: Algorithm Complexity

- Yahoo predicts America's Political Winner 0:00-1:25
- Function Abstraction (Explanation of Functions and Algorithms) 1:28-2:45
- What is IN a Spec 2:45-3:30
- What is NOT in a Spec 3:30-5:15
- Reference Text "Introduction to Algorithms" 5:18
- Algorithm Analysis: The Basics** Good for Classroom Instruction 6:00-7:40***
- Algorithm Analysis: Running Time** Good for Classroom Instruction 7:41-8:25**
- Algorithm Analysis: Runtime Analysis Problem and Solution 8:25-9:55
- Runtime Analysis: Input Size and Efficiency 9:58-11:25
- Runtime Analysis: Worst of Avg Case 11:25-13:20
- Run Time: Final Assessment 13:20-16:46
- Example:Finding a student by ID (detailed explanation of input/output) 17:00-31:20
- Ex: Finding a shared birthday (Constant, Logarithmic, Linear, Quadratic, Exponential) 31:21-33:30
- Ex: Finding Subsets 33:40 to End

Accommodations/Differentiation

- Check for food allergies before performing this exercise. If any student has allergies that would put them at risk, substitute another food item or simulate the process with stand-in ingredients.
- The most common allergy will be to peanuts. Possible alternatives in this case include cream cheese & jelly, toast with butter and jam, or even a deli sandwich (turkey/ham/roast beef) with mayo and/or mustard.
- For other allergies, or if no options are acceptable, simulate using fake ingredients. (Even slips of paper with the ingredients written on them can suffice.)

- Students who have previous programming experience may tend to dominate the algorithm generation process. Encourage these students to avoid pointing out errors directly and help the other members of their group find and fix errors.
- If students are struggling with the level of specificity required, allow them to make some basic assumptions to ease the process.
- This can lead to an excellent conversation about abstraction.
- In the "debugging" round, some students may go overboard with the level of detail in an attempt to resolve all possible ambiguities. Remind these students that there are some basic instructions that can be easily understood by most people, and there is no need to go into further detail in those cases.
- If you feel students can handle the discussion, you can draw a parallel to machine code here.

Forum discussion

Lesson 0.2 Algorithms (TEALS Discourse account required).

TEALS Unit 0 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 16, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Algorithm	A complete, well-defined sequence of steps for completing a task or solving a problem.
Computer	An electronic machine that can solve different problems, process data, store & retrieve data and perform calculations.
Computer Science	The study of the principles and use of computers.
Computer Program	A sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem.
Debug	A process of locating and removing computer program bugs, fixing errors or abnormalities.
Programming Language	A vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

Lesson 0.3: Programming Languages

Learning Objectives

Students will be able to…

- Complete levels in the game LightBot 2.0
- Complete small programs in SNAP with guidance
- Explain why computer programs are written in specialized languages

Materials/Preparation

- ☐ Classroom computers with internet access
- ☐ **IMPORTANT: At least a few days prior to class,** ensure that the classroom computers can load the website for both activities. If not, work with school IT to solve the problem.
  - If you are not able to load SNAP on your classroom computers, you will not be able to proceed with the course. Test this well ahead of time and make sure your school's IT staff knows what the requirements

are and can help achieve them.
- ☐ Work through both the LightBot Hour of Code activity and the SNAP Hour of Code activity on your own so you are familiar with the activities and can provide assistance as needed
- ☐ Part 3 of the LightBot activity calls itself “Loops” but is really using recursion (specifically tail-recursion). The exercises are still valuable, but be prepared to potentially discuss, or at least point out, the distinction so students are not confused when they encounter normal loops later.
- ☐ Unit 0 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
5 minutes	Introductory discussion
35 minutes	Lightbot/SNAP activities
10 minutes	Debrief and wrap-up

Instructor’s Notes

1. Introductory discussion

- Introduce students to the concept of a **Computer program**: a sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem
- Ask for any programming languages students are familiar with (even just names).
- Draw distinctions between proper programming languages and other types of languages (such as markup languages e.g. HTML).
- Lead the group to develop expectations about what aspects of programming might be most challenging and what skills might be most useful to be successful.
- Sample guiding questions:
- What are the steps required to write a computer program:
- This is essentially developing an algorithm for writing a program!
- What knowledge might make writing a program easier?
- What might you need to do when writing a computer program that you have never or rarely done before?
- What parts of programming are most intimidating or scary?
- What are you good at that might help you be a good programmer?

2. Activity

- Students should work through the LightBot 2.0 Hour of Code activity and/or the SNAP Hour of Code Activity.
- Choose one activity as the requirement, and leave the other for those who finish quickly.
- Allow students to struggle with the activities if needed, stressing the importance of patience and persistence in programming.

3. Debrief

- Guide students in a discussion about the activities. Some sample guiding questions:
  - What was most challenging?
  - Put special emphasis on the iterative nature of programming, and the need to occasionally throw out a partial solution and start over.
  - What was different about solving these problems than solving other problems encountered in school?
  - Why can instructions not be given in simple English? Why must we be limited to certain operations from which we must build up solutions?
  - If students seem interested, this can be an opportunity for a brief conversation about the difference between high-level programming languages and machine languages (assembly code).

BJC Lecture Suggestions

OOP Ex: Sketch Pad Dr. Ivan Sutherland "Father of Computer Graphics 15:45-22:10

Background Information for Instructors

BJC Video Suggestion: BJC Lecture 5: Programming Paradigms

- Dilemma of Being a Cyborg 0:00-2:30
- Programming Paradigms 2:30-3:50
- Snap! BYOB (Hybrid) 3:55-4:45
- Functional Programming (Cascading Values) 4:50-5:35
- Imperative/Sequential 5:41-8:35
- Object Oriented Programming (OOP Basic Explanation) 8:40-15:45
- OOP Ex: Sketch Pad Dr. Ivan Sutherland "Father of Computer Graphics *Good for Classroom Instruction 15:45-22:10
- OOP in BYOB (Demo of Functions in BYOB) 22:35-29:20
- Declarative Programming 29-22-31:20
- Declarative Programming Examples in BYOB 31:25-35:20
- Review of Paradigms 35:25-end

Accommodations/Differentiation

- Avoid telling students directly how to solve problems in the activities, and instead encourage them to try many approaches and build on partial solutions.
- This will get them accustomed to the iterative nature (and natural frustrations) of programming.
- If students appear to be frustrated enough to not be enjoying the activity, provide a step in the right direction and remind them that programmers rarely create a correct program on the first try.
- Students who appear frustrated to the point of disengaging can be allowed to skip certain steps or work with a partner, but should NOT be able to opt out of the entire activity.
- If there is only enough time for one activity, choose based on the following guidance:
- LightBot is more challenging, but not substantially so, and is more game-like, which often leads to greater engagement. LightBot is recommended if students seem capable of handling the challenge.
- The SNAP activity is simpler, and most of the material is covered in the first few lessons of Unit 1, so it can be skipped without losing any experience. However, in a suspected high-needs classroom, the easier goals and extra time with SNAP may be beneficial.
- It is unlikely that students will finish *both* activities in one class period. On the rare occasion some do, encourage them to explore SNAP on their own or to try the full version of LightBot 2.0

Forum discussion

Lesson 0.3 Programming Languages (TEALS Discourse account required).

TEALS Unit 0 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 16, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Algorithm	A complete, well-defined sequence of steps for completing a task or solving a problem.
Computer	An electronic machine that can solve different problems, process data, store & retrieve data and perform calculations.

Word	Definition
Computer Science	The study of the principles and use of computers.
Computer Program	A sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem.
Debug	A process of locating and removing computer program bugs, fixing errors or abnormalities.
Programming Language	A vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

Lesson 0.4: SNAP Self-Portrait

Learning Objectives

Students will be able to...

- Create a simple "program" in SNAP to describe themselves

Materials/Preparation

- ☐ Do Now: I am ______
- ☐ Lab 0.4 handout (Getting to Know You) (Download in Word) (Link to PDF)
- ☐ Student Experience Survey (Download in Word) (Link to PDF)
- ☐ Optional: digital camera
- ☐ Unit 0 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
5 minutes	Introduce Activity
35 minutes	Getting to Know You lab
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Introduce Activity

- Tell students that today they will explore SNAP and use it to create a "self-portrait" program.
- Emphasize that the goal of today's lesson is ***not*** for students to develop a deep understanding of any of the features in SNAP. Later lessons will teach them everything they need to know. For now, they should just explore, figure out what they can, and put it to use however they see fit.
- Spend just a couple minutes demonstrating how to open SNAP, create sprites and scripts, and run programs.
- **DO NOT** go into detail– the specifics of SNAP will be covered in much more depth in unit 1. The goal here is simply to give students enough of a starting point to be able to explore and try things out on their own.
- Show students the lab handout and read through the instructions.
- Point out the places to write answers to the written questions in parts 1.2 and 1.3.
- Draw special attention to the list of requirements for the self-portrait program in part 1.4.
- This is an excellent opportunity to tell students that all labs in this course will look similar to this, and that they should get used to reading instructions carefully.
- The lab handout incoprates the *Student Experience Survey* questions avaialble in the Classroom Plan appendix. It is a great starting point to learning about the cultural references of your students, but you can and should go beyond that and integrate opportunities to share and learn about your students in class activities and assignments. In this lab students write a short program that shares an aspect of their cultural background or interests; teachers can create a model to show the class as a demo. There is also a question asking students

to share stories or sayings from their family. You could use sayings provided (with student permission) to decorate the classroom and reinforce classroom learning values.

2. Activity

- Instruct students to complete the “Getting to Know You” lab.
- For part 1.3, you can either assign pairs or allow students to pair up on their own. If students are allowed to choose their own partners, take care to ensure that no one is excluded. If you choose to assign pairs, try to partner students into groups that are heterogenous by ability or background. Try to pair students who do not normally interact.
- If the class has an odd number, form a single group of three.
- Optional
- If you have a digital camera available, you can take photos of the students and have them use their picture as the costume to the sprite. Doing so will give a level of personalization to each student’s project. You will need to provide students with a location to download their photo. In addition you need to walk students through the process of importing a costume from the file menu.

3. Debrief

- Ask each student to identify and describe *one* feature they discovered in SNAP. Keep a running list on the whiteboard or projector.
- If the students build a pretty comprehensive list, you can use this as a chance to go over a brief roadmap for the course.
- Ask students what they enjoyed about working with SNAP and what they disliked.

BJC Lecture Suggestions

BJC Video Suggestion: BJC Lecture 4: Functions

- Types of Blocks 18:15-19:45

Accommodation/Differentiation

- Allow students to use whatever resources they need, including instructors, peers, and web searches if necessary, to create their programs. The goal of this lesson is exploration and exposure, not mastery.
- For students that are hesitant or unwilling to engage, point out some simple, useful features (like basic drawing and the Say Block block) to get them started. Remind them that complexity or “coolness” are not important for this lab.
- If you wish, you can add a second day to this lesson for students to share out their programs on a volunteer basis. Work with your classroom teacher and/or school IT staff to determine the best way for students to be able to present.
- One simple option is to have volunteering students share their projects with you, and then show each one in turn on the teacher computer/projector.
- Have a plan for collecting or accumulating the students’ programs, which you can then use to get to know your class!

Forum discussion

Lesson 0.4 SNAP Self Portrait (TEALS Discourse account required).

TEALS Unit 0 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 16, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Algorithm	A complete, well-defined sequence of steps for completing a task or solving a problem.
Computer	An electronic machine that can solve different problems, process data, store & retrieve data and perform calculations.
Computer Science	The study of the principles and use of computers.
Computer Program	A sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem.
Debug	A process of locating and removing computer program bugs, fixing errors or abnormalities.
Programming Language	A vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.

Do Now 0.04 I am

How would you fill in the following: I am ________?

This could be answered in many ways:

1. What you enjoy doing: I am a musician.
2. Physically: I am tall.
3. Personal characteristic: I am funny.
4. Career aspirations: I am an engineer.
5. Self reflective: I am a deep thinker.
6. As an athlete: I am a runner.

And this is just the beginning.

After taking few minutes to think who you are, write down a few. We will be using who you are to write a Snap self expression program.

Lab 0.4 - Getting to Know You

In this lab, you will explore SNAP and create a simple "self-portrait" program to introduce yourself to your instructors and classmates.

Part 1 - Exploring SNAP

1. Open SNAP on your computer and spend a few minutes looking around, trying things out, and seeing what the language can do. Don't worry about understanding everything completely– we'll go through things in a lot more detail soon. Just try to get a sense of some of the basic capabilities.
2. Write down three things you found that SNAP can do and how to do them. Be as specific as you can!

SNAP can do this...	If I do this...
1.	
2.	
3.	

3. Find a partner and compare notes. Share your findings with your partner and ask him or her about what he or she learned. Write down the three capabilities your partner found below.

SNAP can do this...	If I do this...
1.	
2.	
3.	.

Part 2 - Who Are You

Using what you've learned and other things you might discover, create a SNAP program that describes yourself. This can take whatever form and use whatever SNAP tools and blocks you want. Make sure that somehow, at some point in your program you show the information from the Student Experiences Survey. (Download in Word) (Link to PDF)

Grading Scheme/Rubric

Lab 0.4 Criteria	
1.2 Listed 3 things you found Snap can do	0.3 points
1.3 Listed 3 things your partner found Snap can do	0.3 points
Part 2	
Includes name	0.2 points
Current grade	0.2 points
Age	0.2 points
Programming or computer experience	0.0 points
Learning CS going to be like	0.2 points
Favorite uses of technology	0.2 points
Favorite subject in school	0.2 points
Example of something learned	0.2 points
Hobby or interest	0.2 points
Family Saying	0.2 points
PROJECT TOTAL	**2.4 points**

Unit 1

Lesson 1.1: Welcome to SNAP

Learning Objectives

Students will be able to...

- Define and identify "blocks," "scripts," "sprites," and "the stage" in SNAP.
- Write simple SNAP programs
- Describe what simple SNAP programs do without executing the code

Materials/Preparation

- ☐ Unit 1 Do Now (Link to Do Now Handout)
- ☐ Lab 1.1 handout (Download in Word) (Link to PDF)
- ☐ Helping Trios handout Helping Trios handout (Download in Word) (Link to PDF)
- ☐ Ensure that all classroom computers can access http://snap.berkeley.edu/run.
- ☐ Read through the lab so that you are familiar with the requirements and can assist students as needed
- ☐ Unit 1 Tips

Pacing Guide

Duration	Description
Day 1	
10 minutes	Welcome, attendance, Do Now, announcements
10 minutes	Introductory discussion
10 minutes	Lab walkthrough
20 minutes	"Welcome to SNAP!" Lab activity
5 minutes	Demonstrate turn-in procedures and wrap-up
Day 2	
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review from yesterday
30 minutes	Continue lab
10 minutes	Debrief, turn-in, and wrap-up

Using Zoom Blocks

Zoom Blocks

Zoom Blocks are a useful tool to increase the readability of code in Snap. To access the Zoom Blocks feature, simply go up to settings in the upper left, and select the second option in the list, Zoom Blocks. Once selected, an interface pops up which allows you to increase the zoom on your code and shows you a preview.

Instructor's Notes

Day 1

1. Introductory discussion

- Review the definitions of "algorithm" and "program" developed in lesson 0.2
  - **Algorithm:** a complete, well-defined sequence of steps for completing a task or solving a problem
  - **Program:** a sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem

2. Activity Walkthrough

- Work through Lab 1.1 up through part 3 (scripts) as a class.
- Point out key aspects of SNAP in each section, including:
  - the sections of the window in part 1
  - block shapes and color-coding of categories in part 2
  - simply draw attention to the different shapes at this point; their meanings will be covered as each block type is introduced
  - the "drag-and-drop" nature of the language
  - running blocks/scripts by clicking on them
- Ensure that all students are able to create an account and save their work, as describe in the preamble of the lab
- Allow students a few minutes to develop their own answers to each of the questions and activities before discussing as a group

3. Activity

- Individually or in pairs, have students continue working through the "Welcome to SNAP" lab activity.
  - If you choose to assign pairs, try to partner students into groups that are heterogenous by ability or background.
  - Try to pair students who do not normally interact.
  - Students should answer all questions and complete all activities and turn them in using your chosen turn-in procedure.
- For written questions, either have students hand-write answers and turn in the hard copies or set up an electronic submission system of some kind.

- For SNAP programs, including the Kaleidoscope program, students should save the program to the cloud and share a link with you
- Students should aim to get through at least part 6 by the end of Day 1
- Throughout the period, you can pause class to discuss each numbered part of the lab before moving on
- Circulate while students are working and try to judge when the majority of the class has finished each part
- Try to check in at least every 10 minutes

4. Turn-in procedures

- Demonstrate the turn-in procedure you will use for student work throughout the semester, and have students follow along to turn in their work from the lab.
- Ensure that each student is able to turn in their work before the class period ends.

Day 2

1. Review

- Go over answers to the questions from the parts of the lab completed on day 1 (ideally, at least through part 6)
- Include the parts completed as a class (parts 1-3)
- Ask questions along the way to assess students' understanding of concepts.

- Conisder the following questions:
  - Naming the parts of the SNAP window
  - Defining "block," "reporter," "script," etc.
  - Describing the coordinate system used in SNAP

2. Continue lab

- Students should continue working through the lab, aiming to finish all parts by the end of the day
- As before, students should turn in all answers using your chosen turn-in procedure.
- Pause class at least once to verify understanding of parts 7 and 8 before students move on to the Kaleidoscope program
- Judge the appropriate time based on observing student progress, but ensure that you break in with at least 10 minutes remaining so students have enough time to work through the program

3. Debrief and wrap-up

- Discuss the challenges in the Kaleidoscope program
- Ask students how the challenges were similar to or different from those they encountered when playing LightBot in Lesson 0.2
- If time allows, ask one or two students to demonstrate their programs and describe their code
- Remind students of the turn-in procedures discussed yesterday and ensure all students are able to turn in their work

BJC Lecture Suggestions

BJC Video Suggestion

- BJC Lecture 2: 3D Graphics
- Procedural Based Motion (Lab 1.1): 16:56-20:00

Accommodations/Differentiation

- For students that finish the lab early, encourage them to add more advanced features to their Kaleidoscope program, exploring parts of SNAP not covered in the lab.
- Students that are struggling with the lab can be paired up and/or receive individual instructor attention to help them through the activity. You can also use Helping Trios.

- No parts of this lab can be easily skipped without impacting learning objectives, so provide as much support or scaffolding as you can to ensure all students are able to complete the lab. Add days to the lesson if needed.

Forum discussion

Lesson 1.1 Welcome to SNAP (TEALS Discourse account required).

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many “frames” or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Do Now 1.1 Cartesian Coordinate Plane Review

Using the graph paper provided, draw the Cartesian Coordinate plane. Include the following,

- x-axis
- y-axis
- Origin
- x - Minimum = -15
- y - Minimum = -15
- x - Maximum = 15
- y - Maximum = 15

Connect the dots

Using the Coordinates below connect the points to see what they create. Hint: Have you seen Red?

Curve 1 Begins; (-5, -3); (-4, -5); (-3, -7); (-3, -6); (-2, -4); (-1, 0); Curve Ends

Curve 2 Begins; (-7, 1); (-8, 4); (-9, 7); (-8, 10); (-7, 10); (-5, 9); (-3, 7); (-2, 6); (0, 7); (2, 6); (3, 7); (5, 9); (7, 10); (8, 10); (9, 7); (8, 4); (7, 1); (3, 4); (0, 1); (-3, 4); (-7, 1); Curve Ends

Curve 3 Begins; (-5, 0); (-4, 0); (-2, -1); (-4, -1); (-5, 0); Curve Ends

Curve 4 Begins; (5, -3); (4, -5); (3, -7); Curve Ends

Curve 5 Begins; (7, 1); (8, -3); (9, -7); (8. -7); (7. -6); (4, -9); (1, -10); (-1, -10); (-4, -9); (-7, -6); (-8, -7); (-9, -6); (-8, -2); (-7, 1); Curve Ends

Line 6 Begins; (5, 0); (4, -1); (2, -1); (4, 0); (5, 0); Curve Ends

Curve 7 Begins; (1, 0); (2, -4); (3, -6); (3, -7); (1, -8); Curve Ends

Curve 8 Begins; (-4, -1); (-4, 0); (-3, -1); Shade In Area; Curve Ends

Curve 9 Begins; (-3, -7); (-1, -8); (1, -8); (2, -6); (1, -5); (-1, -5); (-2, -6); (-1, -8); Shade In Area; Curve Ends

Curve 10 Begins; (4, 6); (7, 9); (7, 5); (6, 4); (4, 5); (4, 6); Shade In Area; Curve Ends

Curve 11 Begins; (4, -1); (4, 0); (3, -1); Shade In Area; Curve Ends

Curve 12 Begins; (-7, 4); (-7, 9); (-4, 6); (-4, 5); (-6, 3); (-7, 4); Shade In Area; Curve Ends

Example Including Curve 1

Coordinate Starter Image

Lab 1.1 - Welcome To SNAP!

SNAP is a programming language, which you can use to tell a computer what to do. A program is a particular set of instructions for the computer to follow.

Programs in most languages use only letters (and punctuation), but SNAP is different: it's a visual language. Instead of writing a program only using the keyboard, you will drag pictures of blocks and click them together.

The following is a program in SNAP!:

Simple Program in Snap!

Can you guess what it might do? (Write your guess below)

SNAP is different than many other languages in another way— you run it in a web browser like Firefox or Chrome. The URL that you can use to always get to SNAP! is

http://snap.berkeley.edu/run

In order to save your programs, the first thing you'll need to do is make an account. In the SNAP browser window, Find the cloud-shaped button in the top toolbar on the upper left corner of the window:

Cloud Button

Click it, select the "sign up" option in the menu, and follow the instructions there. You will need to check your email after creating your account to get your initial password.

Cloud Sign Up

1. Overview of the Window

You may have noticed that there are a few main sections of the SNAP! window. These regions are named as shown below.

Snap! Overview

2. Blocks

The area at the left edge of the window is the palette. As you see in the picture, it contains tabs for eight different-color block categories. In this lab, we will focus on the Motion, Sound, Pen, and Sensing tabs. You will learn about the other tabs in the next few labs.

These tabs are an important organizational structure in SNAP because they are home to the various blocks that you will use to tell the computer what to do. The blocks are categorized under each tab based on what kind of thing each block does.

2.1) Below, fill in the name of the category to which each block belongs.

Block	Category
Play Note	
Clear	
Mouse x	
Touching	
Change y by	
Distance to	
Point in direction	
Stop All Sounds	

Look at the **Motion** tab. Under this tab you will find a bunch of blocks that correspond to motion-like actions. For example, click on the Move block, drag it to the scripting area, and drop it anywhere in the scripting area.

Move block dragged onto the stage

The block that you just dragged and dropped into the scripting area controls something that we call a sprite, which is the arrowhead-looking thing in the middle of the stage (the white part of the window).

Back to the scripting area, if you click on the move 10 steps block you just put there, the sprite will move 10 steps. You can see this visually depicted by the sprite moving in the stage. You can vary the input of the block, i.e., the number 10, to change the number of steps you want to the sprite to move.

2.2) How can you change the block input so that the sprite moves in the opposite direction?

Part 3. Scripts

Now that you have figured out how to make a sprite move, you might be wondering how to make the sprite do other things as well. To make a sprite do more than just move, we need to use different types of blocks and link them together. You can link blocks by Snapping (hence the name SNAP) them together – drag a block right underneath the one to which you want to attach it. Blocks will SNAP together when one block's indentation is near the tab of the one above it. You should see a white bar appear like the one in the image below, which just shows you where the block will go after you drop it.

Two blocks about to SNAP together

If you keep attaching blocks together in this way, you will create a script. A SNAP program consists of one or more of these scripts.

3.1) Try recreating the following script in the scripting area in SNAP.

Script with move and say blocks

The purple say… blocks are available from the **Looks** tab.

Remember, a script will tell the sprite what to do. Click on the script and see what happens! You will know that your script is running if it has a highlighted border around it:

script with highlighted border

3.2) What happens when you run this script?

Script with move and say blocks

Be sure to note: **blocks in a script run in a specific order, from the top of the script to the bottom**. Generally, SNAP waits until one block has finished its job before continuing on to the block below it. (One common exception is blocks that play sounds: a block's job can be to start the sound, which means the block below it will execute while the sound is still playing.

Part 4: Reporters

At the bottom of Motion palette are three blocks shaped differently from the others. The oval-shaped x position and y position are called *reporters*. (We don't need the third one right now.) Unlike the jigsaw-puzzle-piece-shaped command blocks we've used until now, reporters don't carry out an action (such as moving the sprite or displaying a speech balloon) by themselves. Instead they report a value, usually for use in another block's input slot.

These particular reporters tell you where the sprite is on the stage. As in algebra class, **x** means left-to-right position, and **y** means bottom-to-top position.

Drag your sprite to the far right side of the stage. Next, drag an x position block into the scripting area and click on it. You should see a little speech balloon next to the block:

x position reporting

4.1) What value does the x position block report to you when the sprite is?

...at the far right side of the stage: ...in the center of the stage: ...at the far left side of the stage:

Click on the gray box to the left of the **x position block** in the palette, and then look over to the stage. You will see that the value that the block would report is displayed on the stage:

x position checkbox

x position watcher

This on-stage display is called a *watcher*.

The x position and y position the will tell you the position of your sprite on the screen. Move the sprite around and the values reported by these blocks change.

Part 5: Position On The Stage

A sprite occupies a position (x,y) on the stage where x represents the horizontal position, from -240 (left) to 240 (right), and y represents the vertical position, from -180 (bottom) to 180 (top). Here's a picture:

x-y grid

The black sprite is at the center of the stage, called the origin, with coordinates (0, 0). The green sprite is to the right of the origin, so its x position is positive. The green sprite is also below the origin, so its y position is negative. Each grid line above represents 20 steps, so the green sprite's coordinates are (140, -100). Take some time to make sure you understand this; discuss it with a classmate.

5.1) What are the coordinates of the red sprite?

In your *SNAP!* window, take a look at the blocks under the Motion tab. The majority of the blocks there will help you position your sprite on the stage. Try them and see what they do! Change the input values to see what happens.

5.2) List at least 4 blocks from the Motion tab that will change the position of a sprite.

Part 6: Experiment with Drawing Commands

Try to get comfortable with the blocks under the Motion tab and the Pen tab. Figure out what each one does and try to use these blocks to draw a square or a simple picture.

6.1) What do these blocks do? (write an explanation next to each block)?

move 10 steps

turn 15 degrees

clear

pen up

pen down

**6.2) Does the *turn* block change the sprite's x and/or y position?

**6.3) Using these blocks, draw a square. Write the code (blocks) you used below.

Tips and Tricks:

Once the pen is down, it stays down even in a different script. Use the pen up block to lift the pen so that no lines will be drawn.

You also will want to show the direction and x and y position of the sprite. In the Motion tab, you can select for these to be shown on the stage as described in the reporters activity you saw earlier in the lab.

Part 7: Follow that Mouse

forever go to mouse x-y

7.1) What do you think the script above will do?

Hint: mouse x and mouse y are reporters in the Sensing palette; they tell you where the mouse is pointing.

Copy the code into SNAP, and click on the `forever` block to run it.

Did it follow your expectations (Yes/No)?

7.2) What happens when you drag the mouse to a different part of the screen while the program is running?

7.3) How does program's behavior change when you modify the `go to` block as shown below?

go to mouse x + 30, y

Part 8: Forever and a Day

From the previous exercise, you may have figured out what the forever block does. The `forever` block is the first block you have seen that holds, or wraps around, other blocks. We call this a *C block* because of its shape. As the name `forever` implies, it will run the blocks inside it again and again and again and … well, forever. You will find this block under the **Control** tab.

Will a forever block ever stop?

Not unless you tell it to: Click on the stop sign icon on the upper right hand corner of the SNAP! window.

stop button

This stop sign will stop all scripts that are running in any sprite. This is equivalent to executing the stop all in the Control palette.

Check for Understanding

8.1) How many times will the sprite say "Hello"?

forever say

a) 1
b) 2
c) 10
d) continuously

8.2) Assuming the sprite starts in the middle of the stage and pointing in direction 90, where would it end up after running this script?

forever move

a) Farther right on the stage
b) Farther left on the stage
c) Off the stage to the right
d) Off the stage to the left

8.3) What would appear on the screen when this script is run?

forever say animals

a) The sprite would say "Tiger" forever

b) The sprite would say “Tiger” then “Panda” once
c) The sprite would alternate between saying “Tiger” and “Panda” forever
d) The sprite would say “Tiger” and “Panda” at the same time forever.

8.4) Assuming the sprite started in the middle of the stage facing right, what kind of drawing would the sprite make?

forever draw something

a) a circle
b) a dot
c) a cylinder
d) a straight line

Part 9: Make a Kaleidoscope

Explore this drawing program for a little bit (https://aka.ms/kaleidodraw2). Press the spacebar to run the program, and move your mouse cursor over the stage of the SNAP! window. While over the stage, use the **d** (pen down), **u** (pen up), and **c** (clear) keyboard keys to change what gets drawn on the screen. The script that causes the sprite to follow the pointer is

forever go to mouse x-y

As you can see, this drawing program features more **Control** blocks, in addition to the `forever` block first introduced in the *Follow the Mouse activity*. These *hat*-shaped block, which can be used only at the beginning of a script, indicate when a specific script should be run.

Kaleidoscope Activity

kaleidoscope examples

The kaleidoscope consists of 4 sprites. Each sprite will be drawing with a different pen color. Each sprite’s movement is based on the movement of the mouse. The first sprite follows the mouse, just like in the example we looked at before. The other 3 sprites move around as the mouse moves, but reflected over the X and Y axes.

Don’t forget to save and submit your work!

Some tips:

- You will need four sprites. (We haven’t used more than one sprite up to now, but having more than one allows for more interesting projects, as you’ll see.) The easiest way to create three more is to *duplicate* the one you have. Right-click the sprite in the sprite corral, and select **duplicate** from the *context menu* that appears. Each duplicated sprite will have exactly the same scripts as the original, which is why we suggest duplication rather than just creating more sprites from scratch.
- You can change the color of each sprite by clicking the color input in that sprite’s set pen color block (found under the **Pen** tab), choosing a color, and then clicking on the block itself (to run the block and actually set the color). Don’t worry about matching the colors in the animation exactly!
- Pay close attention to what each of the other sprites is doing in the animation above. You will need to modify the **x** and **y** inputs in each sprite’s go to x-y block using simple formulas, with addition and subtraction

    Hint: All the sprites are reflecting in different ways around the (x=0, y=0) origin point of the stage.**

- Once you figured this out, try out some complicated formulas and/or more sprites, and share with your classmates

Grading Scheme/Rubric

Lab 1.1 Criteria	
1.1 What does it do?	0.2 points

Lab 1.1 Criteria	
2.1 Categories	0.4 points
2.2 Move in opposite direction	0.1 points
3.2 What happens	0.2 points
4.1 x positions	0.3 points
5.1 Coordinates of red sprite	0.2 points
5.2 List 4 change position Motion blocks	0.2 points
6.1 What it does	0.3 points
6.2 Does turn block change x or y position	0.1 points
6.3 Draw a square	0.4 points
7.1 What does it do	0.2 points
7.2 Dragging the mouse	0.1 points
7.3 Program behavior w/modification	0.1 points
Total	**2.8 points**
Checking for Understanding	
8 Multiple choice	0.4 points
Total	**0.4 points**
Mini-project	
9 Make a Kaleidoscope	1.0 points
Total	**1.0 points**
PROJECT TOTAL	**4.2 points**

Lesson 1.1: Welcome to SNAP

Learning Objectives

Students will be able to...

- Define and identify "blocks," "scripts," "sprites," and "the stage" in SNAP.
- Write simple SNAP programs
- Describe what simple SNAP programs do without executing the code

Materials/Preparation

- ☐ Unit 1 Do Now (Link to Do Now Handout)
- ☐ Lab 1.1 handout (Download in Word) (Link to PDF)
- ☐ Helping Trios handout Helping Trios handout (Download in Word) (Link to PDF)
- ☐ Ensure that all classroom computers can access http://snap.berkeley.edu/run.
- ☐ Read through the lab so that you are familiar with the requirements and can assist students as needed
- ☐ Unit 1 Tips

Pacing Guide

Duration	Description
Day 1	
10 minutes	Welcome, attendance, Do Now, announcements
10 minutes	Introductory discussion
10 minutes	Lab walkthrough
20 minutes	"Welcome to SNAP!" Lab activity
5 minutes	Demonstrate turn-in procedures and wrap-up
Day 2	
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review from yesterday
30 minutes	Continue lab
10 minutes	Debrief, turn-in, and wrap-up

Using Zoom Blocks

Zoom Blocks

Zoom Blocks are a useful tool to increase the readability of code in Snap. To access the Zoom Blocks feature, simply go up to settings in the upper left, and select the second option in the list, Zoom Blocks. Once selected, an interface pops up which allows you to increase the zoom on your code and shows you a preview.

Instructor's Notes

Day 1

1. Introductory discussion

- Review the definitions of "algorithm" and "program" developed in lesson 0.2
  - **Algorithm:** a complete, well-defined sequence of steps for completing a task or solving a problem
  - **Program:** a sequence of instructions or steps, written in a language that can be understood by a computer, that will be used by the computer to complete a task or solve a problem

2. Activity Walkthrough

- Work through Lab 1.1 up through part 3 (scripts) as a class.
- Point out key aspects of SNAP in each section, including:
  - the sections of the window in part 1
  - block shapes and color-coding of categories in part 2
  - simply draw attention to the different shapes at this point; their meanings will be covered as each block type is introduced
  - the "drag-and-drop" nature of the language
  - running blocks/scripts by clicking on them
- Ensure that all students are able to create an account and save their work, as describe in the preamble of the lab
- Allow students a few minutes to develop their own answers to each of the questions and activities before discussing as a group

3. Activity

- Individually or in pairs, have students continue working through the "Welcome to SNAP" lab activity.
  - If you choose to assign pairs, try to partner students into groups that are heterogenous by ability or background.
  - Try to pair students who do not normally interact.
  - Students should answer all questions and complete all activities and turn them in using your chosen turn-in procedure.
- For written questions, either have students hand-write answers and turn in the hard copies or set up an electronic submission system of some kind.
- For SNAP programs, including the Kaleidoscope program, students should save the program to the cloud and share a link with you
- Students should aim to get through at least part 6 by the end of Day 1
- Throughout the period, you can pause class to discuss each numbered part of the lab before moving on
- Circulate while students are working and try to judge when the majority of the class has finished each part
- Try to check in at least every 10 minutes

4. Turn-in procedures

- Demonstrate the turn-in procedure you will use for student work throughout the semester, and have students follow along to turn in their work from the lab.
- Ensure that each student is able to turn in their work before the class period ends.

Day 2

1. Review

- Go over answers to the questions from the parts of the lab completed on day 1 (ideally, at least through part 6)
- Include the parts completed as a class (parts 1-3)
- Ask questions along the way to assess students' understanding of concepts.

- Conisder the following questions:
  - Naming the parts of the SNAP window
  - Defining "block," "reporter," "script," etc.
  - Describing the coordinate system used in SNAP

2. Continue lab

- Students should continue working through the lab, aiming to finish all parts by the end of the day
- As before, students should turn in all answers using your chosen turn-in procedure.
- Pause class at least once to verify understanding of parts 7 and 8 before students move on to the Kaleidoscope program
- Judge the appropriate time based on observing student progress, but ensure that you break in with at least 10 minutes remaining so students have enough time to work through the program

3. Debrief and wrap-up

- Discuss the challenges in the Kaleidoscope program
- Ask students how the challenges were similar to or different from those they encountered when playing LightBot in Lesson 0.2
- If time allows, ask one or two students to demonstrate their programs and describe their code
- Remind students of the turn-in procedures discussed yesterday and ensure all students are able to turn in their work

BJC Lecture Suggestions

BJC Video Suggestion

- BJC Lecture 2: 3D Graphics
- Procedural Based Motion (Lab 1.1): 16:56-20:00

Accommodations/Differentiation

- For students that finish the lab early, encourage them to add more advanced features to their Kaleidoscope program, exploring parts of SNAP not covered in the lab.
- Students that are struggling with the lab can be paired up and/or receive individual instructor attention to help them through the activity. You can also use Helping Trios.
- No parts of this lab can be easily skipped without impacting learning objectives, so provide as much support or scaffolding as you can to ensure all students are able to complete the lab. Add days to the lesson if needed.

Forum discussion

Lesson 1.1 Welcome to SNAP (TEALS Discourse account required).

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many "frames" or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Lesson 1.2: Building Blocks

Learning Objectives

Students will be able to...

- Name the categories of blocks in SNAP and describe what the blocks in each category do
- Describe the function of several common SNAP blocks (see lab for specific blocks)
- Be able to use common blocks to build simple SNAP programs (see lab for specific blocks)

Materials/Preparation

- ☐ Do Now 1.2: Tracing and Debugging If you like you can place a picture of Admiral Grace Hopper or her Mark II notes that show the bug she found in the computer.
- ☐ Lab 1.2 handout (SNAP Scavenger Hunt) (Download in Word) (Link to PDF)
- ☐ Helping Trios handout Download in Word (Link to PDF)
- ☐ Read through the handout so that you are familiar with the requirements and can assist students
- ☐ Index cards with each students name on them. One name per index card.
- ☐ Unit 1 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introductory discussion
25 minutes	"SNAP Scavenger Hunt" Lab activity
15 minutes	Debrief and wrap-up

Instructor's Notes

Introductory discussion

- Do Now Debrief:
- Tell the story of Admiral Grace Hopper who originated the term "Debugging"
- Review the basics of SNAP from the previous lesson
- Ensure that students can:
  - Define "block" and "script"
  - Describe how to build a script (snapping blocks together)
  - Explain how scripts are executed (one block at a time in order)
- Introduce the concept of block categories
- Ask students to consider why categories are helpful as opposed to having a simple list of blocks

2. Activity

- Individually or in heterogeneous pairs as described in previous lessons, have students work through the "SNAP Scavenger Hunt" activity
- Students should turn in answers to all questions and SNAP programs for the final problems
- Consider gamifying the worksheet or debrief as an online poll and/or quiz

3. Debrief

- Go through each question or prompt in parts 1 and 2 and ask students to share their answer

  Cold calling alternative: Write each student's name on an index card and shuffle the cards. Next, ask a few "review" questions to set the stage that ask for application (or creativity, or evaluation) and have no single "right answer." Give students 30 to 60 seconds to formulate an answer. You can have them talk in pairs or groups for a few minutes as well if you prefer. It's only after students have been thinking that you take the top card to see who answers. Sometimes you can say something like, "If you don't know, make something up like you would on a mid-term, and then we'll all help you develop better answers." The student takes their best shot, knowing that others will help if necessary. At that point you have lots of options: You can ask a follow-up to the same student, pick another card and have somebody else answer the follow-up, or simply have students volunteer to expand on the first answer.

- Point out the color coding for each category
- Emphasize important details in some of the blocks listed in part 2, such as:
- the Think Block block will leave the bubble on the stage until something else is thought
- the Go to XY block and Glide block blocks require explicit x- and y-coordinates
- Solicit a few student responses for each category in part 2.2 before commenting yourself
- Try to guide the students to discuss with each other and settle on a description for each category with minimal instructor intervention
- Discuss one or two student solutions to each of the SNAP programming problems
- Either solicit volunteers or use cold call alternative
- Point out differences between student solutions and call attention to the fact that there is more than one way to solve a problem

Accommodations/Differentiation

- Colorblind students may not be able to identify the block colors, but can still recognize the organization of categories. Be sensitive to this, but no modifications are likely required.
- Students that are struggling with the lab can be paired up and/or receive individual instructor attention to help them through the activity. Use you could also use "Helping Trios."
- The bonus assignment (3.4) should be used for students who finish quickly, and can be a setup for the Animation Project.

Forum discussion

Lesson 1.2 Building Blocks (TEALS Discourse account required).

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many "frames" or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Do Now 1.2 Tracing and Debugging

Define the following terms. You may use the internet.

1. tracing through code means:
2. debugging means:

Lab 1.2: Snap! Scavenger Hunt

In this lab, you will explore the functionality of some common blocks and where they are located in the palette.

1. Locating common blocks

1.1) Fill in the name of the category to which each block belongs in the chart below. The first one is already filled in for example.

Block	Category
point in direction	*motion*
think	
play notes	
set pen color	
glide	
repeat	
change pen size	
set effect	
go to x-y	
set size	
rest for beats	
point towards	.

2. What does it do

2.1) Describe the function of each block in the chart below. If the block accepts arguments (contains values that you can change), be sure to test out a few different ones to make sure you fully understand what those values mean. The first one is already filled in for example.

Block	Function
point in direction	*Changes the direction that the sprite is facing. The argument indicates the number of degrees the sprite turns clockwise from pointing upwards. When the argument is "90", the sprite points right, and so on.*
think	
play notes	
set pen color	
glide	
repeat	
change pen size	
set effect	
go to x-y	
set size	
rest for beats	
point towards	.

2.2) At this point, you may be noticing some patterns. Use what you've learned from exploring these blocks to answer the questions below about each block category.

a. What do the blocks in the **Motion** category do?

____

b. What do the blocks in the **Looks** category do?

____

c. What do the blocks in the **Sound** category do?

____

d. What do the blocks in the **Pen** category do?

____

3. Put it all together

You are now going to use some of the blocks you've explored to create, save, and submit a Snap! program.

1. Create a script that plays 4 different notes with at least 2 rests in between.
2. Use the repeat block to play your song on loop
3. Create a script that initializes the sprite at position (-20, 10). Then, have the sprite draw a shape that has at least 2 different colors and 2 different line thicknesses. An example would be a square that has 2 thin red sides, and 2 thick blue sides.
4. BONUS: Create a SNAP script that draws a picture of your choice (such as draw a picture that represents your favorite music, food or hobby). Use at least four of the blocks from parts 1 and 2. Be creative!
5. When you've completed all of the scripts above, save your file, share it, and then copy the unique URL below. Be sure to share and publish your file before pasting the URL.

File URL:

Grading Scheme/Rubric

Lab 1.2 Criteria	
1.1 Locating common blocks	0.2 points
2.1 What does it do?	0.4 points
2.2 Categories	0.3 points
4 different notes, 2 rests in between	0.4 points
repeat block plays song	0.3 points

Lab 1.2 Criteria	
multi-color, multi-line thickness shape at (-20,10)	0.4 points
BONUS: draw picture with at least 4 blocks from 1.1 and 2.1	0.5 points
PROJECT TOTAL	**2.5 points**

Lesson 1.3: Drawing Shapes

Learning Objectives

Students will be able to...

- Construct simple algorithms to draw shapes
- Convert algorithms into SNAP programs

Materials/Preparation

- ☐ Do Now 1.3: Drawing a Triangle You can ask students to give instructions in plain language, pseudo code or with images.
- ☐ Lab 1.3 handout (Squares and Triangles and Stars, Oh My!) (Download in Word) (Link to PDF)
- ☐ Geometry Exterior Angles Review
- ☐ Unit 1 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review and introduce activity
25 minutes	Shape drawing activity
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Review

- Review the categories of blocks and what each is for.
- Put particular emphasis on Motion and Drawing, as those will be used for this assignment
- Remind students about the iterative process of programming
- Students may get frustrated throughout this activity; remind them that requiring multiple attempts to find the right solution is normal.

2. Activity

- Students should complete the "Triangles, Squares, and Stars, Oh My!" activity individually.
- Point out that the shapes in part 2.1 are not necessarily listed from easiest to hardest, and that the scripts need not be written in the given order.
- Here is a Geometry Cheat Sheet by Math Salamanders showing various shapes and their respective angles. It can be presented to students as a reference during the activity and/or enlarged and placed at the front of the classroom for use throughout unit 2.
- When students finish, have them turn in their project using whatever procedures you have set up.

3. Debrief

- Have students switch seats with a nearby classmate and review each other's work
- If one in a pair student was able to complete a program and the other was not, have the student who was successful walk his/her partner through

- If neither student in a pair was able to complete a program, encourage them to work together to figure out what they were missing
- Make sure course staff is available to help pairs who cannot figure out a given problem
- Ask students to share what they learned from looking at their partner's programs
- Point out the fact that there is more than one way to solve a problem and two programs that are both correct might not look the same

Accommodations/Differentiation

- Advanced students can attempt the five-pointed star. If students finish that shape, encourage them to try more advanced shapes.
  - Examples: Kite, smiley face, student's initials
- Struggling students can either be paired or allowed to not complete certain shapes.
- Students who have not yet taken geometry may have difficulty determining the correct angles. Point out to them that geometric understanding is not necessary—the angles can be determined using trial and error.
  - If students need more scaffolding, they can be pointed to an online resource such as http://www.mathsisfun.com/geometry/interior-angles-polygons.html
  - If most students are not equipped to figure out the angles on their own, provide diagrams like the one of a square in part 1.1 to assist.

Forum discussion

Lesson 1.3 Drawing Shapes (TEALS Discourse account required)

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many "frames" or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Do Now 1.3: Drawing a Triangle

Setup

Open the starter *Snap!* project, exit full screen, and save it to your account. You can use this as a starting point for today's lab.

> Try enabling "Visible steps" near the middle of the top toolbar. Adjust the slider so you can see what happens as *Snap!* steps through your program.

Visual Steps UI

Instructions

Using only the blocks already added on screen (not every block is needed), create a *Snap!* program that draws an equilateral triangle, like:

Result triangle

Lab 1.3: Squares and Triangles and Stars

In this lab, you will write your first SNAP programs to draw some simple shapes on the stage.

1. Drawing a square

1.1) Write a SNAP script that draws a square when the number 1 is pressed on the keyboard. Remember that each corner of a square is a 90° angle, as shown in the figure below.

square diagram

1.2) Add code so that the sprite says the word “square” while it is drawing. The sprite should stop saying “square” once it has finished drawing the square.

1.3) Add code so that pressing the space bar clears the pen marks from the stage.

2. Adding more shapes

2.1) Now that you’ve drawn a square, add code to draw the shapes in the following table. Each shape should be drawn when the number next to it is pressed on the keyboard. (For example, pressing 3 on the keyboard should draw a diamond.)

When this key is pressed…	Draw a …
1	Square
2	Equilateral triangle
3	Diamond
4	Pentagon
5	Parallelogram (“leaning rectangle”)
6 (optional)	5-pointed star

You may want to draw a diagram similar to the one above of a square to figure out the angles in each shape.

2.2) Add code so that as each shape is being drawn, the sprite is saying the name of shape. The sprite should stop saying the name of the sprite when it is finished drawing.

2.3) Modify your code so that each different shape is drawn in a *different* color and with a *different line thickness*. So, for example, if the square is drawn in blue with a line thickness of 3, each other shape must be drawn in a color that is not blue and with a line thickness that is not 3.

2.4) Add code so that the sprite is hidden when it is not drawing. This is will make sure that the sprite is not obstructing the view of your beautiful artwork!

Grading Scheme/Rubric

Lab 1.3 Criteria	
1 - Square	0.25 points
2 - Equilateral Triangle	0.25 points
3 - Diamond	0.5 points

Lab 1.3 Criteria	
4 - Pentagon	0.5 points
5 - Parallelogram	0.5 points
Star (bonus!)	0.5 points
PROJECT TOTAL	**2.5 points**

Lesson 1.4: Animation

Learning Objectives

Students will be able to...

- Animate SNAP sprites using costume changes and movement
- Trigger action in other sprites using broadcasts

Materials/Preparation

- ☐ Do Now 1.4: Sprite Communication

- ☐ Lab 1.4 handout (Sprites in Action) (Download in Word) (Link to PDF)
- ☐ Unit 1 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Lecture and introduce activity
30 minutes	Animation activity
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Lecture

Note: The Lesson 1.4 project has examples of all the concepts covered in this lesson.

1. Introduce students to the "Costumes" tab and show them how to import costumes to a sprite.
   - Point out that, while a sprite's costumes can be anything, most often the different costumes of a single sprite will be somehow related.
2. Demonstrate how to use the "next costume" and "switch to costume" blocks to change the appearance of a sprite.
   - Emphasize that costumes will cycle and that switching to the costume a sprite is already "wearing" is OK.
   - Combine costume switching with movement and other actions to show that blocks of different categories can be combined in the same script.
3. Introduce the "broadcast" and "when I receive" blocks and show how they can be used to coordinate action between sprites
   - Point out that all sprites "hear" a broadcast and any sprite with a corresponding "when I receive" will react.
   - Discuss the difference between "broadcast" and "broadcast and wait" and ask students to come up with ideas for when each would be useful.

4. It's worthwhile to introduce the rotate buttons located at the top left of the SNAP interface. These buttons allow you to snap (puns!) your rotation, allowing you to lock rotation, allow free rotation, or allow only left/right rotation.

   rotate buttons

2. Activity

- Students should complete the "Sprites in Action" lab individually.
- Ensure that students create two different sprites for parts 1 and 2 so they do not get the costumes mixed up.
- Students can submit the assignment using your turn-in procedures or you can check off the work as students complete it.

3. Debrief

- Ask a student to show their solution to each part. Call on a different student (either a volunteer or via cold calling) for part 1 and part 2.
- Note any places in which there may be multiple possible approaches. Ask for volunteers to describe differences in their code.

Accommodations/Differentiation

- Students that finish quickly can be encouraged to add more detail to their animations, such as sounds, "say" blocks, and/or more costumes.
- Especially advanced students can be encouraged to create their own costumes for a new animation that does not use any built-in sprites.
- With struggling students, re-emphasize the fact that each sprite can have different costumes and that changing costumes is very much like changing position by moving. Get students to be able to change costumes on a key press before moving on to the animation aspect.

Forum discussion

Lesson 1.4 Animation (TEALS Discourse account required)

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many "frames" or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Do Now 1.4 Example Animation

Complete the following steps individually:

1. Open this starter project.
2. Log into your SNAP account and save the starter project so you can refer back to it later. Name the project "Do Now 1.4 - Example Animation".
3. Press the space bar on your keyboard several times in a row. Notice what happens to the two bats on the screen.

   Hint: If a sprite moves completely off the Stage, you can bring the sprite back on the Stage by right-clicking its icon in the Sprite Corral and choosing "show".
4. Use your mouse to point the cursor at one of the bats. Then, click the mouse button several times. Notice what happens to the two bats on the screen.
5. Take a look at all of the parts of the SNAP window. Try to figure out how the project works. Be sure to explore the tabs at the top of the scripting area and the icons in the Sprite Corral.

   Hint: It may be helpful to activate "visual steps" with the footprint icon near the middle of the top toolbar.

Lab 1.4: Sprites in Action

In this lab, you will use costumes and movement to create simple SNAP animations.

1. Run, Spot, Run

1.1) In a SNAP project, click on the file menu, then click costumes. Import the costumes "dog2 a" and "dog2 b".

Menu->Costumes

Import Costumes

1.2) Write a script to make the sprite change costume each time the space bar is pressed. The sprite should switch back and forth between the two costumes.

1.3) Add code so that the sprite will face to the left, move a few steps, and change costume when the left arrow key is pressed. If you've done it right, it should look like the dog is walking when you press the left arrow key repeatedly.

1.4) Add code so that the dog can walk to the right as well.

2. Here be Dragons

2.1) Create a new sprite. Following the same steps as in part 1.1, import the costumes "dragon1-a" and "dragon1-b".

2.2) Write a script to make the sprite appear to breathe fire when the 'f' key is pressed. The sprite should switch to the "fire-breathing" costume for a few seconds, then switch back to the "normal" costume.

2.3) Modify your code so that the dragon "attacks" the mouse pointer when the 'f' key is pressed. When the 'f' key is pressed, the dragon should take the following actions in order:

- Make sure it is in the "normal" costume
- Point at the mouse pointer
- Glide to the mouse pointer's position
- Change to the "fire-breathing" costume
- Pause to breathe fire
- Change back to the "normal" costume

3. Run Away

3.1) Add another sprite to your program. (This sprite can have any costume you choose).

3.2) Write a script to make this new sprite point away from the dragon and move when the 'r' key is pressed. (You'll need more than one block to do this.).

3.3) Modify your code so that instead of moving when the 'r' key is pressed, the new sprite moves when the dragon "attacks." The "fleeing" sprite should move when the dragon starts breathing fire.

3.4) Add a second sprite that runs away from the dragon as well.

Grading Scheme/Rubric

Lab 1.4 Criteria	
2.2 Dragon breathes fire	0.25 points
2.3 Dragon attacks mouse pointer correctly	0.75 points
3.2 A sprite runs away from dragon	0.25 points
3.3 Sprite runs away when dragon breathes fire	0.5 points
3.4 Another sprite runs away too	0.25 points
PROJECT TOTAL	**2.0 points**

Unit 1 Quiz SNAP Basics

Learning Objectives

Formative assessment on student progress: To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Materials/Preparation

- TEALS Classes can access the Quizzes by logging into the TEALS Dashboard and navigating to "Additional Curriculum Materials" -> "Intro CS Curriculum".
- You will need to log in using incognito mode on the browser.
- See Additional Curriculum Resources for further instructions.

Lesson 1.5: Storytelling Project

Learning Objectives

Students will be able to...

- Apply basic programming and SNAP skills to create an animated movie, play, nursery rhyme, or other scene
- Practice good debugging skills to correct issues as they arise while programming

Materials/Preparation

☐ Do Now 1.5: Day 1, Knock Knock

☐ Do Now 1.52: Day 2, Click Event Interaction

☐ Do Now 1.53: Day 3, Movement Interaction

☐ Reference to Storytelling project: Project 1 - Storytelling (Download in Word) (Link to PDF)
☐ A list of possible plays to find a scene to animate: Most popular Plays and Musicals

- ☐ A list of possible nursery rhymes
  - Wikipedia has a fairly comprehensive list: https://en.wikipedia.org/wiki/List_of_nursery_rhymes
  - A few suggested subjects for students who struggle to come up with their own
  - Pick a few simple, well-known options from your list
- ☐ Planning Worksheet (Download in Word)
- ☐ Unit 1 Tips

Pacing Guide

Duration	*Description*
Day 1	
5 minutes	Welcome, attendance, bell work, announcements
30 minutes	Review unit concepts
20 minutes	Introduce project
Days 2-5	
5 minutes	Welcome, attendance, bell work, announcements
10-15 minutes	Review
30-35 minutes	Lab time
5 minutes	Exit ticket

Instructor's Notes

1. Review

- Play a review game (such as GrudgeBall to remind students of the skills and concepts that have been learned in this unit
- Categories of blocks
  - Movement
  - Drawing
  - Hide/Show
  - Costumes
  - Broadcasting
- Remind students that their solutions to previous assignments are an excellent resource when trying to accomplish similar tasks.

2. Introduce project

- Walk students through project specification, pointing out important details, potential pitfalls, and specific requirements
- The words to the story must appear on the screen somehow
- This can be as a sprite, as a part of the background, or "said" by a narrator
- Sprites must act out the story line by line
- The action must be dynamic– a series of static images is not sufficient
- The action and words should advance automatically, but at a slow enough pace that the viewer can follow
- The user must be able to restart the animation (in a manner other than pressing the green flag again) after it has concluded
- At least two sprites must act during the animation, and the sprites must collectively meet the requirements on page 2
- Encourage students to look at the grading rubric on page two repeatedly throughout the project to ensure they are meeting all the requirements
- Sample project solution

3. Project

- This project is a summative assessment for the unit. Students should be demonstrating mastery of all the skills covered.

- Most students will require roughly 4-8 hours of total work time to complete the project
- Assess the progress of your students regularly using such techniques as asking them to demonstrate their incomplete programs, tracking questions asked during lab time, and/or utilizing peer reviews.
- Adjust the amount of time allowed for the project to fit the needs of your students
- It is vital that nearly all students complete the project before moving on
- If most students have the ability to work on SNAP assignments at home, the amount of in-class time provided can be reduced if necessary.
- If this approach is taken, be sure to make accommodations for students who are *not* able to work at home, such as after school lab hours
- Ensure that students are able to ask questions in class throughout the project
- See the standard Lab Day Lesson for detailed plans for lab days.

Accommodation/Differentiation

- Instead of a scene from a play or a nursery rhyme, students can recreate a famous scene from a movie, depict lyrics from their favorite song, or develop their own unique animations. Feel free to modify the specification for whichever version will appeal most to your students.
- Advanced students can be encouraged to add detail and/or complexity to their project for possible extra credit.
- For ELL students or students from other cultures, offer the opportunity to choose an animation subject familiar to them rather than requiring a traditional English nursery rhyme or play.
- If necessary, a non-English language can be "used for this assignment without affecting the learning objectives. Discuss this option with the student's ELL specialist to determine if it is appropriate.

Forum discussion

Lesson 1.5 Storytelling Project (TEALS Discourse account required).

TEALS Unit 1 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 1, 2, 3, 4, 5, 6, 8, 11, 12, 13, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Blocks	Puzzle-piece shapes that are used to create code in Snap!.
Scripts	Different types of blocks linked together.
Sprites	An object in Snap! which performs functions controlled by scripts.
Stage	The background of a project, performs functions through scripting.
Costume	A costume is one out of possibly many "frames" or alternate appearances of a sprite.
Receive Block	Code Block in Control that receives a message from another block.
Watcher Block	reporter blocks you can click the checkbox for; they will appear in the Stage and you can track them.
X Position	The position that a sprite or the mouse is at along the horizontal axis.
Y Position	The position that a sprite or the mouse is at along the vertical axis.

Do Now 1.5 Day 1 Knock Knock

Write a SNAP program where 1 sprite is telling another a "knock, knock" joke. Internet search for one if you need ideas. Use these blocks:

Broadcast

when I receive

Do Now 1.5 Day 2 Click Event Interaction

Write a SNAP program where 1 sprite acts as a button and when pressed, it increases the size of another sprite. Use these blocks:

Broadcast

when I receive

when I am

change size by

Do Now 1.5 Day 3 Movement Interaction

Click on this starter project.

Look at the code on the Bat's script page and how events are triggered by two different keys. Trace through the scripts to understand how the code works.

Modify the program so that the Bat flies to the Dragon when the space key is pressed ONCE. Program the Bat to say "BOO!" when it stops. Program the Dragon to then turn around say, "You scared me!". Program the Bat to then turn around and fly away.

Project 1: Animated Storytelling

Students will use SNAP basics to implement an animated version of a story.

Overview

Storytelling is a great way to convey culture. Some examples of storytelling are plays and nursery rhymes. Famous plays like those of William Shakespeare have been performed over centuries. Some have been adapted for modern times like West Side Story. A nursery rhyme is a short poem or song written for children. Though the term is typically applied to British or other English language poems, similar concepts exist in many world cultures. These short stories are generally meant to entertain and/or calm young children. Some are believed to have a hidden moral or meaning related to historical events, but many of these meanings are questionable.

Emphasize with students

Digital tools and technologies can help capture stories in our own heritage, especially ones that might otherwise be lost, or difficult to write down In USA there are over 500 Native American communities, speaking more than 290 distinct languages, and a multitude of dialects. Students may be familiar with many states and cities in USA that have been given names adapted from the original Aboriginal language spoken in the area, such as Alaska ("peninsula"), or Minnesota ("cloudy water"), or Seattle ("named after a Native American Chief"). Traditionally Native American people relied on storytelling instead of the written language to pass down information and history.

Many other communities have used storytelling as a method to pass down history. And these stories were often shared from generation to generation by word of mouth from a relative or elder.

For this project, student may be encouraged to portray an animation that depicts some aspect of their own heritage, especially stories that have been passed down by word of mouth. Some ideas:

- animation of how a State/city/town name in USA came to be
- animated map of an immigration journey
- a personal family story

Reference

- Traditional plays
- Nursey rhymes
- History Of Immigration To The US

Details

Behavior

You will create a short animation in SNAP depicting a story of your choice.

- Whenever the green flag is clicked, your SNAP animation should display your chosen story line by line somewhere on the stage. (This should work correctly even if the last run was interrupted and restart.)
- As each line is shown, sprites should act out the story.
- The animation should advance on its own, but should do so at a pace that allows each action to complete and the viewer to read the line before the next line is shown and new action begins.
- In addition, the sprites must act out the story; you should not simply create a series of static backgrounds or costumes that show a stop*motion version of the story.
- Each line must be readable and must stay shown while the corresponding action is occurring.
- When the story ends, there should be a way for the user to replay the entire animation from the beginning.
- You are free to be as creative as you like with your choice of sprites and actions.

You may choose from the sprites provided by SNAP or create your own. (You will not be graded on your artistic skills.) You may interpret the story literally or be clever with your depiction (but don't go too far). However, all sprites, behaviors, words, and animations must be school*appropriate.

If you choose a particularly long story, you may not need to animate the whole thing. Please check with your teacher if you think your idea is long enough for this.

Implementation Details

1. Fill out a Planning Worksheet for the above program. Make sure you consider all aspects of the program carefully.

2. As mentioned above, your animation must display the text and animate each line. Action must be performed by sprites and must consist of more than simply changing costumes. You must include the following components in your animation:

- At least two sprites that act in some way to contribute to the depiction of the story
- At least one sprite that moves
- At least one sprite that rotates
- At least one sprite that changes costume
- At least one sprite that is both hidden and shown at some point

Note that multiple of these requirements may be satisfied by the same sprite (e.g. the same sprite can both move and change costume), but you must have at least two separate sprites that act in the animation.

Sharing

Stories are meant to be shared. Prepare to demo your animation with a partner, in front of the whole class, or with your family members. See if your audience can understand the meaning of your animation, and be prepared to provide some background information associated with the story. The animation can also be video captured and shared online.

Emphasize with students (Continued)

Curriculum Competencies * Design Sharing

As you create software, you will need to keep the end*user, or final audience, in mind. Be thinking of what you are hoping to achieve or communicate when you are creating a piece of work, and be prepared to explain your thoughts behind the ideas.

Grading Scheme/Rubric

Functional Correctness (Behavior)	
Animation depicts a story	2 points
Story is shown one line at a time	2 points
Each line is accompanied by sprites depicting the story, and all action is related to the current line	3 points
Clicking green flag starts animation from beginning	1 point
Animation progresses at a reasonable pace	2 points
User is able to restart animation when it concludes	2 points
Total	**12 points**
Technical Correctness (Implementation)	
Program shows good creativity and effort	2 points
At least two sprites participate in the action	2 points
At least one sprite moves	1 points
At least one sprite rotates	1 points
At least one sprite changes costume	1 points
At least one sprite hides and/or appears	1 points
Total	**8 points**
PROJECT TOTAL	**20 points**

Unit 2

Lesson 2.1: Loops

Learning Objectives

Students will be able to...

- ☐ Define "loop" in a programming context
- ☐ Explain why loops are useful
- ☐ Implement simple repeat and forever loops in SNAP
- ☐ Utilize loops to reduce redundancy in code

Materials/Preparation

- Unit 2 Tips
- Do Now 2.1: Intro Loops
- Lab 2.1 handout - Squares and Triangles Redux (Download in MS Word) (Link to PDF)
- Geometry Cheat Sheet __ [Starter Code]

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and examples
25 minutes	Activity - Squares and Triangles
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Lecture

1. Introduce and discuss concepts of code redundancy and readability

- Remind students that a program can be written in many different ways that are functionally equivalent.
- Ask students to speculate as to why one version of a program might be better or worse.
- Possible answers: more efficient (in time or space), shorter code, more elegant/readable code
- Use this example to demonstrate unreadable code
- Show students the code, ask what it does, then ask if they can think of ways to improve it
- Attempt to get students to realize that the code is *redundant* and could be simplified if there were a way to execute a block of code more than once

2. Introduce loops

- Begin with general definition: *A type of block that causes other code to run multiple times in succession*
- Use real life loops to introduce the concept- water cycle, eating one spoonful at a time,use a poem like "Still I Rise" by Maya Angelou or a song with a repetitive hook like "Happy" by Pharrell Williams. If you choose to use a song, you can break students into groups and have each group choose their own song. Make sure to ask students to identify a song that has a repetitive hook without explicit lyrics.
- Introduce SNAP specific loops:
- Repeat Block runs the body of the loop the specified number of times
- Number of iterations can be a value, variable, or reporter
- Forever Block runs the body of the loop nonstop until the script is ended
- Can be stopped either by clicking the stop sign or by any version of Stop Block
- Repeat Untill Blocl runs the body of the loop until the specified condition becomes true
- Save detailed discussion of this loop until conditionals are introduced

3. Walk through examples of Repeat Block and Foreve Block

- Emphasize usefulness in reducing redundancy and complexity, especially for repetitive tasks
- Simple examples are here

2. Activity

- Direct students to complete "Squares and Triangles Redux" individually.

- If available, students should use their solutions to Lab 1.3 ("Squares and Triangles and Stars, Oh My!") as a starting point. Ensure students "Save as…" before starting on the new lab to not overwrite their original project (part 1.1).
- If student solutions for Lab 1.3 are not available, or are not correct, provide a correct implementation (the solution to Lab 1.3 can be found on the TEALS Dashboard under Additional Curriculum Materials).
- Encourage students to try to use as few blocks and have as little code duplication as possible to draw each shape while still creating understandable scripts. Asa reminder you may want to make the Geometry Cheat Sheet showing various shapes and their respective angles as a reference for students throught unit 2.
- Once students complete part 2.1, the remaining parts should go much more quickly as they all follow the same basic pattern.

3. Debrief

- Discuss one or two student solutions to part 2.2
- Ask students to think about what the code would look like without loops
- Discuss one or two students solutions to part 3.1
- Point out how unwieldy the code for these two shapes would be without loops

Accommodation/Differentiation

- More advanced students can add additional shapes, including a five-pointed star without interior lines.

- Particularly advanced students can be encouraged to build pictures by combining multiple shapes (e.g. a house built of squares of various sizes).

Sample Solution

Lab 2.1 Solution

Forum discussion

Lesson 2.1 Loops (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.1 Intro Loops

Go to this starter project, run the code and test it, then answer these questions in a discussion comment below.

1. What is the forever block, and why is it important for this code?
2. What happens if you take it out temporarily, reattach the rest of the code to the "When Green Flag clicked" block, and run/test the code?
3. What does "point direction 90" do to the sprite?
4. What would happen if you decrease or increase the # of steps?
5. Add code so that if the user presses the left arrow key, the sprite faces the left direction and moves a few steps in that direction. Test to make sure that this works and also that right arrow key still works.

Lab 2.1 - Squares and Triangles Redux

In this lab, you will rewrite your SNAP programs from Lab 1.3 to draw shapes using loops.

Section 1 - Back In Time

1. Open your solution to the original "Squares and Triangles and Stars, Oh My!" activity. Go to the "File" menu and select "Save as..." to give your project a new name. Save as

Section 2 - Simplifying Code

1. Look at your code to draw a square. It is probably quite long and has lots of repeated blocks. Using what you have learned about loops, rewrite this script to be shorter and have less redundancy. Make sure that your code still works as originally intended.
2. Now modify your other shape scripts to also use loops. In all cases, try to have as few blocks and as little redundancy as possible while still keeping your code easy to read and understand.

Section 3 - Adding More Shapes

1. Add code to your program to draw the extra shapes below. Follow all the original guidelines (different color and line thickness for each shape, say the name while drawing) and use loops to keep your scripts as short as possible.

When this key is pressed...	*Draw a...*
7	Decagon (10-sided polygon)
8	Circle

You may not be able to draw a true circle, but you should get as close as you can.

Grading Scheme/Rubric

Lab 2.1 Criteria	
2.2 Square using loop	0.5 points
Section 3	
Decagon	0.5 points
Circle	0.5 points
Minimum redundancy in all	0.5 points
PROJECT TOTAL	**2.0 points**

Lesson 2.2: Nested Loops

Learning Objectives

Students will be able to...

- Use nested loops to solve programming problems

Materials/Preparation

☐ Do Now 2.2: Debugging Loops
☐ Lab 2.2 handout - Yellow Brick Road (Download in Word) (Link to PDF)
☐ Unit 2 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Review, lecture, and examples
25 minutes	Activity - Yellow Brick Road
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Review

- Ask to students to restate the definition of "loop"
- *A type of block that causes other code to run multiple times in succession*
- Ask students to explain why loops are useful in programming
  - To reduce code redundancy and increase readability when dealing with repetitive tasks
- Solicit examples of problems that can be solved with loops
  - Drawing polygons, repeating an action, etc.
- Ask students how their scripts to draw shapes improved when they introduced loops
  - Code became shorter and more readable
  - Small changes, such as altering the size of the shape, became simpler

2. Lecture

1. Ask students to consider the problem of drawing two squares next to each other
   - Work through writing a script to do this. The likely result will duplicate the code to draw a single square:

     twe Squares Example Code

   - Remind students to remember what they previously learned and use loops
   - Once the script is complete, ask students if there are ways they could improve the code
2. Discuss what would happen if you needed to draw 10 squares, or 100, or an unknown number.
   - If students seem capable, you can allude to user input for the unknown number example, but do not get into details at this point
   - Ask if loops can somehow be applied to reduce redundancy even further in the script to draw two squares
3. Introduce the concept of nested loops
   - Define **nested loop** - a loop used inside the body of another loop.
   - Point out that there is nothing particularly special happening here– the body of a loop can contain any code, including another loop
   - Emphasize that the inner loop will run all its iterations *each time* the outer loop runs.
4. Walk through rewriting the script to draw multiple squares to use nested loops:

   Two Squares Example code

   - Ask how many total times the sprite will move 50 steps
   - The sprite will move 50 steps 8 times (4 * 2)
   - Show that the number of squares drawn can be easily changed by simply changing the number of times the outer loop iterates

3. Activity

- Direct students to complete the Yellow Brick Road activity individually or in pairs. Try to pair students that have not previously interacted.

- Remind students to continue using the principles learned with basic loops
- Encourage students to write scripts that are as short and succinct as possible while still being functional and readable

4. Debrief

- Show a student's solution
- Either request a volunteer or cold call alternative
- Point out uses of nested loops
- Ask students to describe how loops, specifically nested loops, made the assignment easier
- Each brick, or at least each row, would have needed to be coded separately creating much longer scripts
- Ask students to think about what code would need to change to alter the size of each brick or the size of the road and how that was made easier with loops
- Size of road: number of iterations in one or both loops
- Size of brick: one or both move blocks **and** number of iterations in one or both loops
- Without loops, the change would have been needed in many different places

Accommodation/Differentiation

- Not all students will recognize all the possible places to use nested loops in the final program. Encourage students to find as many places to use a nested loop as possible, but allow struggling students to focus on just one or two uses.
- Advanced students can be encouraged to change the size of the bricks, which will require not only altering the number of steps moved but also the number if iterations in the loops. You can also ask students to draw a building/house using only squares. For example, they can be instructed to draw where they live, favorite building, or school.

Sample Solution

Lab 2.2 Solution

Forum discussion

Lesson 2.2 Nested Loops (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.

Word	Definition
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.2 Zigzag

Follow the link to a DoNow22 puzzle.

Reassemble the blocks so Alonzo asks how many zigzags to draw. Then have Alonzo draw zigzags from the center of the stage to the upper right the number of times specified:

Alonzo drawing zigzags

Lab 2.2 - Yellow Brick Road

In this lab, you will use nested loops to draw a yellow brick road using as little code as possible.

Part 1 - Brick by Brick

1. Write a SNAP script to draw a single 20x10 "brick" in the lower left corner of the stage when the green flag is clicked.
2. Modify your code to draw two bricks side by side. The bricks should share a short edge, like this: Two yellow bricks
3. Now modify your code again to build a full row of bricks across the entire length of the stage. Use a loop to keep your code as concise as possible. Remember that the stage is 480 pixels wide.

Part 2 - Build a Road

1. Now that you can build a row of bricks, add code to build a second row above the first row. The bricks in the second row should share a long edge with those in the first row, but should be "offset" so that the ends of the second row bricks are at the middle of the first row bricks, like this: Offset yellow bricks
2. Modify your code to build four total rows, alternating back and forth between the "regular" and "offset" rows. Use nested loops to keep your code concise.
3. Finish off the road the by building alternating rows all the way to the top of the stage. Your final road should look like this: Brick Road

Grading Scheme/Rubric

Lab 2.2 Criteria	
1.2 2 bricks bottom left	0.5 points
1.3 row on bottom	0.5 points
2.1 second row offset	0.5 points
3.3 complete road	0.5 points
No extra bricks	0.25 points
Uses at least one loop	0.5 points
Minimum redundancy in all	0.25 points
PROJECT TOTAL	**3.0 points**

Lesson 2.3: Inputs and Conditionals

Student Objectives

Students will be able to…

- Ask for and receive user input in a SNAP program
- Use simple conditional (if and if-else) blocks to alter control flow in a SNAP program

Materials/Preparation

- ☐ Do Now 2.3: Stairs
- ☐ Lab 2.3 Handout - What Shape is That? (Download in Word) (Link to PDF)
- ☐ Unit 2 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
20 minutes	Lecture
20 minutes	Activity - What Shape is That?
10 minutes	Debrief and wrap-up

Note: This lesson may cover too much material for some classes to handle in one day. Feel free to spill over into a second day, splitting the material however works best for your class.

Instructor's Notes

1. Lecture

1. Point out that, so far, our programs have had minimal interactivity
    - There was *some* user interaction in the shape drawing labs, but no back and forth– just pushing a key to trigger an action
    - This is not normally how computer programs work
    - Ask students for examples of interactive computer programs
    - Ask students for suggestions for making previous labs or activities more interactive
    - There are no right or wrong answers here– you're just trying to get students thinking about interactivity
2. Introduce the Ask Block block
    - Demonstrate that it functions similar to the Say Block block but waits for a response from the user
    - The response is stored in the Answer Block block and can be referenced later
    - Variables will not be introduced until unit 3, so this block will be somewhat magical for now. You should judge your students' preparedness to handle the details and act accordingly.
    - Emphasize that only one input is stored at a time, and that asking a new question deletes the previous answer
    - For example, the following script, intended to draw a bunch of squares where the user specifies both the size and the number of squares, will not work as intended:

      Draw Squares Example Code
3. Introduce conditional statements

- Define **conditional** - a block used to make a choice between executing two different chunks of code
- You can also use this video on conditionals by Flocabulary.
- Point out the differences between If Block and If Else Block

- Namely, if-else provides a choice between two code paths, whereas if simply chooses between executing code or not
- Emphasize that **only one** of the bodies, either the if or the else, will ever be executed
- Show students the relational operators (<, >, and =)
- These should be fairly intuitive to most students

2. Activity

- Direct students to complete the What Shape is That? activity individually or in pairs.
- Help students realize that, although they may seem quite different, parts 2.1 and 2.2 require very similar code

3. Debrief

- Discuss one or two students solutions
- Point out differences between the approaches of different students and lead discussion about advantages and disadvantages
- Place particular emphasis on the choice between if and if-else blocks
- Explain that, when conditions are mutually exclusive (as in part 2.1), a series of if vs. if-else blocks can be functionally equivalent
- When the conditions are not mutually exclusive (as in part 2.2), the choice matters more

BJC Lecture Suggestions

Good for Classroom Instruction

- BJC Lecture 13: Mislababled as 14) Human-Computer Interaction Bjorn Hartman
- Why Study User Interfaces Ex:Mouse Xy axis, Sketchpad, PC, Tablets 15:00-25:00
- Example Project: Using Dexterity for Computer Interface Video 28:30-29:30
- Multi Touch Apps and Toolkits 32:00-End

Background Information for Instructors BJC Lecture 13: Mislababled as 14) Human-Computer Interaction Bjorn Hartman

- Bjorn Hartman Background 0:00-3:30
- Human Computer Interface(HCI) 3:45-6:00
- HCI: Design, Computer Science, Applied Psychology 6:00-8:00
- Iterative Design Cycle 8:00-10:30
- Understanding Users 10:35-11:35
- Prototype Interface Examples 11:40-14:00
- Evaluation (Formative, Summative) 14:50
- Why Study User Interfaces Ex:Mouse Xy axis, Sketchpad, PC, Tablets 15:00-25:00
- What had changed? Research: Mainframe to Ubiquitous Computing 25:00-29:30
- Example Project: Using Dexterity for Computer Interface Video 28:30-29:30
- Zipf/Power Law Distribution 30:00-32:00
- HCI Research at Berkeley 32:10-46:25
- Multi Touch Apps and Toolkits 32:00-End

Accommodation/Differentiation

- Advanced students can be encouraged to add extra functionality, such as attempting to draw the shape the user is specifying or identifying types of triangles (equilateral, isosceles, scalene)
- Students who are struggling can be allowed to skip some parts of the tables in 2.1 and 2.2, focusing on just a few cases

Sample Solution

Lab 2.3 Solution

Forum discussion

Lesson 2.3 Inputs and Conditionals (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.3 Stairs

1. Use the following "Repeat" block to draw a square (note: you will have to put a number in place of the blank!).

   Repeat

2. How would you use an additional "Repeat" block to draw 12 squares in a line one next to each other. The each square will be to the right of the last square drawn.
3. How would you modify the code so the squares form a set of stairs going up?

Lab 2.3 - What Shape is That

In this lab, you will use user input and conditional statements to identify shapes based on the number of sides and some other properties as given by the user.

Part 1 - Triangle...or no triangle

1. Write a SNAP program that asks the user to think of a shape and input how many sides it has. Then, if the user is thinking of a triangle, tell him or her so. Regardless of whether or not the shape was a triangle, thank the user for playing.
2. Modify your program to give an appropriate message both when the user *is* thinking of a triangle and when he or she is *not* thinking of a triangle. Give the same thank you message afterward in either case.

Part 2 - Name That Polygon

You will now expand your program from above so that instead of just deciding if a shape is a triangle or not, your program will be able to name the specific polygon. Your program must be able to identify at least the following shapes:

Number of sides	*Polygon name*
3	Triangle
4	Quadrilateral
5	Pentagon
6	Hexagon
any other number	Unknown

1. Fill out a Planning Worksheet for the above program. Make sure you consider all aspects of the program carefully.
2. Write the program. Be sure that your program works correctly in all cases.

Part 3 - Quadrilateral Fever

Quadrilaterals come in many different varieties. Add code to your program so that, if the user is thinking of is a quadrilateral, you ask more questions to find out which type of quadrilateral it is. Your program should give the most specific name that applies. The following table describes the quadrilaterals you should know about from most to least specific:

Property	*Quadrilateral name*
All four sides have the same length and all four angles have the same measure	Square
Not a square and all four angles have the same measure	Rectangle
Not a rectangle and each side is parallel to one other side	Parallelogram
Not a parallelogram and two sides are parallel to each other	Trapezoid
Not a trapezoid	Unknown quadrilateral

Quadrilaterals Sets

1. Fill out a Planning Worksheet for the above program. Make sure you consider all aspects of the program carefully.
2. Write the program. Be sure that your program works correctly in all cases.

Grading Scheme/Rubric

Lab 2.3 Criteria	
1 Identifies triangle	0.5 points
2 Identifies polygons	1.0 points
3 Identifies quadrilaterals	0.5 points
PROJECT TOTAL	**2.0 points**

Lesson 2.4: Variables

Learning Objectives

Students will be able to...

- Use variables to track values throughout a program

Materials/Preparation

- ☐ Do Now 2.4: Debugging
- ☐ Lab 2.4 handout - Guessing Game (Download in Word) (Link to PDF)
- ☐ "The Box Variable Activity" materials (Optional)
- ☐ Unit 2 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and introduce activity
25 minutes	Activity - Guessing Game
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Lecture

1. Review user input, specifically the Answer Block block
    - Ask students to speculate about how the Answer Block block works
    - Students should recognize that the block must be storing a value somehow and remembering it for later
    - Ask students whether that type of functionality might be useful in other cases
2. Introduce variables
    - Define and explain the concept of a **variable:** *a location in memory to store a value for retrieval and use later*
    - Consider introducing variables with an interactive physical demonstration by modifying the The Box Variable Activity for your students.
    - Demonstrate creating, assigning, and accessing a variable in SNAP **Ignore global vs. this sprite only for now**
    - Point out that a variable can only hold one value at a time
    - When a new value is assigned, the old value is lost and cannot be recovered
    - Emphasize the importance of descriptive, readable names for variables
    - Show that variables are independent
    - One variable's value can be assigned to another, as in Set Var1 to var2 Block, but changing the value of `var2` later will not update the value of `var1`

2. Activity

- Briefly demonstrate the Pick Random 1 to 10 block block, which will be used in the lab
- Direct students to complete Guessing Game individually or in pairs

3. Debrief

- Discuss and demonstrate one or more students' submissions
- Ask students for commentary on usage and naming of variables throughout program

Accommodation/Differentiation

- Advanced students can be encouraged to implement statistics (best score, average guesses/game, number of time each secret number chosen, etc.)
- Struggling students can be given code that completes part 1.1, and possibly also part 1.2, to get them started
- Students who are particularly overwhelmed should focus only on parts 1.2 and 2.2

Forum discussion

Lesson 2.4 Variables (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.4 Debugging

Follow the link to a Two Sprites Tag project with a few bugs.

Explain what the bugs are and how to fix them. There are THREE bugs.

Lab 2.4 - Guessing Game

In this lab, you will use conditional statements and variables to build a simple number guessing game.

Section 1 - I'm Thinking of a Number

You will write a SNAP program to choose a random number between 1 and 10 and then ask the user to guess a number. If the user's guess matches the random number, the user wins. If not, the user loses. In either case, the user should be shown a message indicating whether they won or lost and the secret random number should be revealed.

1. Fill out a Planning Worksheet for the above program.
2. Write the simple version of the guessing game program described above.
3. Modify the program to keep asking the user for guesses until the correct number is given. Be sure to give a message after each guess, but only reveal the secret number when the user has guessed correctly and the game is over.

4. Add code to ask the player their name at the start of the game. Then, personalize the message for an incorrect guess by adding the player's name. For example, if Sarah is playing the game, then the message should say "Sorry, Sarah, that guess is not correct" instead of just "Sorry" when Sarah guesses incorrectly.

Section 2 - Game Upgrades

1. Modify your guessing game so that the player can decide the range of possible numbers from which the secret number can be chosen. After asking the player's name, ask what they want the highest possible number to be. Then, instead of choosing a random number between 1 and 10, choose a random number between 1 and the number the player requested.
2. Add code to keep track of how many guesses the player has made. After the player guesses correctly, inform them how many tries it took before the correct number was guessed.
3. Increase the player's chances by telling them whether the guessed number is too high or too low instead of just that it is incorrect.

Grading Scheme/Rubric

Lab 2.4 Criteria	
1.1 Planning Worksheet	0.5 points
1.2 Simple version	0.25 points
1.3 Repeat till correct guess	0.25 points
1.4 User name	0.25 points
2.1 Range of numbers	0.25 points
2.2 Number of guesses	0.25 points
2.3 High or low	0.25 points
PROJECT TOTAL	**2.0 points**

Lesson 2.5: Boole in the House

Learning Objectives

Students will be able to...

- Define and identify Boolean expressions and operators
- Evaluate Boolean expressions
- Utilize Boolean operators (and/or/not) to create compound conditions

Materials/Preparation

- ☐ Do Now 2.5: Variables Practice
- ☐ Lab 2.5 handout (Triangles of All Kinds) (Download in Word) (Link to PDF)
- ☐ Unit 2 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
20 minutes	Review and lecture
20 minutes	Triangles activity
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Review

- Remind students about conditional statements, and ask what the blocks that can go in the holes in if blocks have in common
- Blocks are "pointy" and all are "yes/no" or "true/false" questions

2. Lecture

- Introduce Booleans with the "Stand Up, Sit Down:" Warm Up activity of the Code.org "Booleans Unplugged" lesson
- Define **Boolean expressions** as expressions that evaluate to true or false
  - If desired, explain that the term "Boolean" is derived from George Boole
- In SNAP, all Boolean expressions are pointy six-sided blocks
- Present the three Boolean operators: and, or and not
- Define the operators and describe when each will return true
- Show the truth tables for each operator and explain how to read them (see below for truth table example)
- Truth tables are simply one way of expressing how the Boolean operators work; if students are struggling, other depictions (such as an exhaustive list of all possible results) can be substituted
- Emphasize that since Boolean operators are themselves Boolean expressions, they can be nested
- Practice evaluating Boolean expressions, starting simple and moving to more complex nested operations
- Start with simple expressions: e.g. `5 < 7 AND 4 > 2`
- Introduce variables: e.g. `x = 7; x < 5 OR x > 10`
- Nest operations: e.g. `(x = 4; y = -3; x == y OR (x > 0 AND y < 0))`
- Discuss short-circuiting in evaluation of Boolean expressions
- Discuss situations in which the Boolean operators might be needed
- The lack of <= and >= operators in SNAP makes for a great example

3. Activity

- Students should complete the "Triangles of All Kinds" activity individually or in diverse pairs (students who have not interacted previously, by ability, etc.)
- A number of geometric concepts (Triangle Inequality Theorem, Pythagorean Theorem, etc.) are used in this lab, but students need not have a deep understanding of them. The necessary points are explained in the lab.
- Encourage students to think about whether an "and" or an "or" is appropriate in each case. Draw out truth tables if necessary.
- As done previously in the unit, you can make the Geometry Cheatsheet available to students.

4. Debrief

- Walk through a student's response
- Point out uses of Boolean operators
- Discuss how nested or chained if blocks could potentially be used to obtain the same behavior, but would result in longer, less-readable code

Logical AND truth table

p	q	p and q
true	true	true
true	false	false
false	true	false
false	false	false

Logical OR truth table

p	q	p or q
true	true	true
true	false	true
false	true	true
false	false	false

Logical NOT truth table

p	not p
true	false
false	true

Accommodations/Differentiation

- Students that have not taken Geometry made be intimidated by some of the concepts in the lab. Deep understanding of the theorems is not necessary; encourage the students to simply focus on the equations and inequalities presented.
- If the students continue to struggle, help them build the necessary expressions, but encourage them to assemble them into the full condition on their own.
- Advanced students, especially those who have taken higher levels of math, can be encouraged to add additional functionality, such as using Heron's formula to calculate the triangles area or using trigonometry to attempt to draw the triangle.
- Drawing the triangle is very challenging.

Forum discussion

Lesson 2.5 Boole in the House (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.

Word	Definition
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.5 Variables Practice

1. Write an algorithm, in English, for how you would swap the values in two variables.

   For example, suppose we have two variables that have different values:

   set [playerOneValue] to “Torch”

   set [playerTwoValue] to “Gold”

   How would you swap the values so [playerOneValue is] set to “Gold” and [playerTwoValue] is set to “Torch”?

   Here, “Torch” and “Gold” are just examples – suppose the algorithm doesn’t know what values the variables have before it begins.
2. Swap your algorithm with a partner and test it to determine if it works or not.

Lab 2.5 - Triangles of All Kinds

In this lab, you will use Boolean operators to determine what sort of triangle a user is describing.

Is that even possible

1. Write a SNAP program that asks the user for the lengths of all three sides of a triangle. Store each length in a variable. Then say the perimeter of the triangle with those three side lengths
2. Add code to your program to check whether or not the three side lengths can form a real triangle. Remember that, in any real triangle, the sum of the lengths of any two sides is greater than the length of the third side. So, if the triangle has side lengths *a*, *b*, and *c*, then all of the following must be true:

   ```
   a + b > c

   a + c > b

   b + c > a
   ```

If the sides given cannot make a real triangle, say so instead of saying the perimeter.

What kind of triangle

1. Add code to your program to determine and say whether or not the triangle described is a right triangle. If the triangle has side lengths *a*, *b*, and *c*, then the triangle is a right triangle if the following is true:

   ```
   a * a + b * b = c * c
   ```
2. Add code to your program to determine which type of triangle has the side lengths given. A triangle can be one of the following three types:

Triangle type	Description
Equilateral	All three side lengths equal
Isosceles	Two side lengths equal, one different
Scalene	All three side lengths different

Grading Scheme/Rubric

Lab 2.5 Criteria	
1.2 Computes perimeter or says invalid	1.0 points
2.1 Identifies right	0.25 points
Section 2.2	
Identifies equilateral	0.25 points
Identifies isosceles	0.25 points
Identifies scalene	0.25 points
PROJECT TOTAL	**2.0 points**

Lesson 2.6: Combining Variables

Learning Objectives

Students will be able to...

- Combine loops with conditionals to create models with repeated but conditional behavior

Materials/Preparation

- ☐ Do Now 2.6: Boolean Practice
- ☐ Lab 2.6 handout (What Goes Up...) (Download in Word) (Link to PDF)
- ☐ Unit 2 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review, lecture and introduce activity
30 minutes	Gravity activity
10 minutes	Review and wrap-up

Instructor's Notes

1. Review and lecture

Review loops and conditionals

- Ask students what loops do, when they are useful, and what loops exist in SNAP
  - loops cause code to execute multiple times
  - loops can help reduce redundancy and increase readability
  - SNAP contains three loops: repeat, forever, and repeat unti
- Ask students what conditionals are for and when they are useful
  - conditionals are used to execute a block of code only under certain circumstances
  - Encourage discussion about previous activities
  - Fill in understanding gaps when necessary

Demonstrate combining loops and conditionals

- Present Repeat Until Block and Forever Block
- Ask students to suggest when these constructs might be useful
  - "repeat until" is useful when a loop needs to run not for a set number of iterations, but until some situation occurs
  - "forever if" is useful when code should execute *any time* a condition is true, for the duration of the program

    - Point out that the condition in “repeat until” is a termination condition, while in “forever-if” it is a continuation condition

Introduce the concept of modeling

- **Modeling:** building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details_
- Ask students to suggest systems or concepts that might need to be modeled
- Discuss important considerations when designing and implementing a model
    - Lead students to realize that most sacrifice some amount of accuracy or realism for simplicity

2. Activity

- Students should complete “What Goes Up…” lab individually
- Mention that the code written in this lab will be helpful for the Platform Game project

3. Debrief

Accommodations/Differentiation

- Take care to ensure that all students have functional code by the end of the lab to avoid putting some students at a disadvantage starting the project
- If many students struggle, consider releasing your own solution after the lab has been completed in class
- Utilize TAs and advanced students to assist struggling students—it is vital that all students complete this lab in advance of starting the project
- Advanced students can consider increasing the realism of their gravity model by adding acceleration and/or beginning to implement jumping

Forum discussion

Lesson 2.6 Combining Loops and Conditionals (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code.“If” statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.

Word	Definition
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.6 Boolean Practice

Open the following Startercode

There are three rooms: Start, Checkpoint and End.

Sonic should be able to move to each room by moving left and right. Start <–> Checkpoint <–> End

Find the errors and complete the code!

Lab 2.6 - What Goes Up

In this lab, you will use everything you've learned about loops and conditionals to construct a simple model for gravity.

Before You Start

Go to the starter project at https://aka.ms/intro-lab2-6. Log into SNAP and save your own copy of the project by choosing "Save as" from the file menu. Be sure to click the "Share" button in the Save dialog box. Highlight the URL in the address bar and copy it.

Channeling Newton

Write a script for the Dino sprite so that:

1. (1 point) When the green flag is clicked.
2. (1 point) Dino goes to the top of the stage.
3. (2 points) Dino falls towards the bottom of the stage in a forever loop.
4. (3 points) But when Dino hits the ground (in other words, if Dino is touching the ground), it should stop falling. You can use the fact that the ground is all the same color green, along with the "touching color" block. You can change the color by clicking on the color, then clicking on anything that has the desired color.

...Must Come Down

1. (2 points) Modify your code so that when the green flag is clicked, Dino will not only move to the top of the stage, it will move to a random x-coordinate on the stage (between -240 and 240). Use the "pick random" block:
2. (2 points) Modify your code so that the Dino sprite not only stops falling when it touches the ground, but it also stops falling if it is touching the Platform sprite. In other words, it should always fall unless it is touching the ground OR it is touching the Platform sprite.

Jump Up

1. (2 points) Modify your code so that when the spacebar is pressed, Dino will jump up. Have Dino jump by using a repeat loop with a block in it that will move Dino up. Dino will have to move up faster than you think since it will be fighting the forever loop that causes it to always go down.

Challenge

1. (2 points) Dino jumps any time the spacebar is pressed, even if it is in midair. How do you make it so Dino can only jump while on the ground or the platform?
2. (2 points) Can you add code so Dino will move right and left with the arrow keys?

3. (5 points) In reality, gravity pulls down with constant acceleration, not constant speed. How can you change your code to model gravity as a constant acceleration instead of a constant speed?

Unit 2 Quiz Loops

Learning Objectives

Formative assessment on student progress: To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Materials/Preparation

- TEALS Classes can access the Quizzes by logging into the TEALS Dashboard and navigating to "Additional Curriculum Materials" -> "Intro CS Curriculum".
- You will need to log in using incognito mode on the browser.
- See Additional Curriculum Resources for further instructions.

Lesson 2.7: Pong Project

Learning Objectives

Students will be able to...

- Implement a well-written version of Pong
- Practice good style and conventions to create readable and maintainable code

Materials/Preparation

- ☐ Do Now 2.7: Bouncing Ball

- ☐ Reference to the Pong project: Pong (Download in Word) (Link to PDF)
- ☐ Pong planning worksheet: (Download in Word) (Link to PDF)
- ☐ [Optional] Printouts of the specification
- ☐ Link to an online version of Pong for demonstration: http://www.ponggame.org/
- ☐ Unit 2 Tips

Pacing Guide

Duration	Description
Day 1	
5 minutes	Welcome, attendance, bell work, announcements
30 minutes	Review unit concepts
20 minutes	Introduce project
Days 2-9	
5 minutes	Welcome, attendance, bell work, announcements
10-15 minutes	Review
30-35 minutes	Lab time
5 minutes	Exit ticket

Instructor's Notes

1. Review/Introduction

- Play a review game (such as GrudgeBall) to remind students of the skills and concepts have been learned in this unit.
  - Variables
  - Operators
- Remind students that their solutions to previous assignments are an excellent resource when trying to accomplish similar tasks.

2. Introduce project

- Walk students through the project specification, pointing out important details, potential pitfalls, and requirements.
- If students are unfamiliar with the concept of the pong game, spend a couple minutes demonstrating one for the class. If you have a SNAP or Scratch version, that works best, but an online game will work as well.

3. Project

- This project is a summative assessment for the unit. Students should be demonstrating mastery of all the skills covered.
- Most students will require roughly 6-10 hours of total work time to complete the project
- Assess the progress of your students regularly using such techniques as asking them to demonstate their incomplete programs, tracking questions asked during lab time, and/or utilizing peer reviews.
- Adjust the amount of time allowed for the project to fit the needs of your students
- It is vital that nearly all students complete the project before moving on
- If most students have the ability to work on SNAP assignments at home, the amount of in-class time provided can be reduced if necessary.
- If this approach is taken, be sure to make accommodations for students who are *not* able to work at home, such as after school lab hours
- Ensure that students are able to ask questions in class throughout the project
- See the standard Lab Day Lesson for detailed plans for lab days.

Accommodations/Differentiation

- If any students do not have the ability to work at home, ensure enough in-class time is provided to complete the assignment, offering extensions if necessary.
- Advanced students can be encouraged to add a single-player mode with a computer-controlled opponent, recreate the original mechanic in which the ball's angle depends on where it hit the paddle, add different "levels" with obstacles or differently shaped fields, or any other extension.
- Struggling students can be exempted from certain features (such as ball speed or scoring) or given starter code
- If students need significant assistance, focus them on the ball's movement, as it is both the most computationally interesting part of the assignment.

Sample Solution

Project 2 Solution

Forum discussion

Lesson 2.2 Pong Project (TEALS Discourse account required).

TEALS Unit 2 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 7, 9, 10, 14, 15, 16, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Loop	A type of block that causes other code to run multiple times in succession. A control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
Nested Loop	A loop used inside the body of another loop.
Conditional	A block used to make a choice between executing two different chunks of code."If" statements.
Variable	A placeholder for some value. Types of variables:Global variables - apply to all sprites Sprite variable - applies to one sprite Script variable - parameter that applies to one script
Boolean Expression	A value (text, number, picture, etc.) that evaluate to true or false.
Boolean Operators	Boolean expressions that can be nested.
Function/Methods	Other programming languages, like Python or Java, use these terms to refer to the same idea as a Custom Block in Snap! a reusable.
Truth Tables	A tool for evaluating the possible inputs and outputs of a Boolean expression.
Modeling	Building a system to simulate the behavior of a real-life phenomenon by simplifying or ignoring certain details.

Do Now 2.7 Bouncing Ball

1. Open a new Snap project and change the sprite's costume to a ball.
2. Using the following blocks, write a script that continuously moves the ball across the screen and bounces the ball when it hits the edge of the screen.

   if on edge, bounce

   when Green Flag clicked

   move 3 steps

   forever
3. Add a second sprite. Add the same script from step 2 above for the new sprite.
4. What happens when you run? Explain what you observe.

Project 2: Pong

Students will implement a well-written and engineered version of the classic arcade game Pong.

Overview

In 1972, when video games were still very new and relatively unknown, a new game took the world by storm. A simple simulation of tennis using two-dimensional graphics, minimal sounds, and extremely basic controls, Pong became the first arcade game to achieve widespread popularity and is credited as the genesis of the modern video game industry. Today, the game has been played, remade, spun-off, and referenced innumerable times and it remains, to many, the single most identifiable and recognizable game in the history of video games.

Details

Behavior

Gameplay Pong is played by two players each controlling a paddle with the goal of defending their end of the "field." A ball begins play in the middle of the screen and, at the start of each round, moves in a random direction. The ball bounces off the upper and lower edges of the field and the players' paddles. Each time the ball bounces off a paddle, its speed increases by a small amount. When the ball bounces off a paddle, its direction is reversed with a small random variation to add unpredictability to the game.

Scoring If the ball touches the left or right edge of the field, a point is scored for the opponent of the player who was defending that edge and the ball resets to the middle of the field. When one player reaches 5 points, the game is over and that player is the winner. The winner is announced on the screen and the players are given the opportunity to start a new game.

Player Control Paddles are positioned a short distance away from the side they are defending, and can only move up and down, not side to side. Each player should have two keys to control the movement of their paddle: one for up, and one for down. Paddles move at a set speed that is the same for both players can cannot be controlled. The player on the left will control his/her paddle with the 'w' and 's' keys. The player on the right will use the up arrow and down arrow keys.

Required Checkpoints

Checkpoint 1 Players can control paddles; the ball starts in the middle, moves in a random direction, and bounces

Checkpoint 2 The ball speeds up when it hits a paddle, and resets to the middle when it hits the left or right edge

Final due date A point is scored when the ball hits the edge on the opponent's side of the field; the game ends when one player reaches five points; players can start a new game after the game ends

Planning Worksheet

Part of the design process is planning. The "Pong Planning Worksheet" is an example of how to plan. Fill out the planning worksheet first before writing any code.

Grading Scheme/Rubric

Functional Correctness (Behavior)	
Players can control paddles with required keys	2 points
Ball begins play at middle of field at start of game and after each point	3 points
Ball bounces correctly off upper and lower edges and paddles	4 points
Ball increases in speed each time it bounces off a paddle	3 points
A point is scored for the opponent each time the ball touches the left or right edge	3 points
Game ends when one player reaches five points	2 points
Winning player is shown when game ends	1 point
Players can begin a new game	1 point
SubTotal	19 points
Technical Correctness (Implementation)	
Gameplay is smooth, polished, and intuitive	3 points
Program shows good creativity and effort	2 points
Program is well-documented and exhibits good style	2 points
Checkpoint 1	4 points
Checkpoint 2	4 points
Sub Total	15 points

Functional Correctness (Behavior)	
Total	34 points

Project 2: Pong Alternative

Students will implement a well-written and engineered alternative to the Pong Project.

Learning Objectives

Students will be able to …

- Research, ideate, and apply personal interests to an application that includes:
    - 1 or more moving sprites
    - 2 or more user keyboard controlled sprites
    - The sprites should interact in a way that allows the users to accumulate a score or value.
- Practice good style and conventions to create readable and maintainable code

Overview

Read over Lesson 2.7 and the Pong Project (project_2.md) file. Your challenge is to create a project of your own that contains similar elements.

Possible Ideas

Jurassic world

The Stegosaurus lived in the Upper Jurassic period around 155 to 145 million years ago. It is one of the most easily recognized dinosaurs, with its distinctive double row of kite-shaped plates on its back, and the long spikes on its tail. The armor was necessary as it lived with such meat-eaters as Allosaurus and Ceratosaurs. Learn more about dinosaur's here:

Imagine that a few Jurassic era dinosaurs are living in Alaska and now being endangered by our changing climate which has caused melting glaciers and sharp icebergs. These beautiful creatures roam around randomly, unaware of impending threats to their survival. Your goal is to move your truck (user-controlled-sprite1) to catch a dinosaur, so they can be safely transported to another habitat where they are safe to roam and survive. Meanwhile a moving iceberg (user-controlled-sprite2) is endangering the animals. When the dinosaur comes in contact with the iceberg, it is injured, and it's movement is affected…

Penny Catcher

The Penny, also called a one-cent piece, is a coin worth one one-hundredth of a dollar. Learn more about the Penny here:

Imagine that the Penny has been discontinued and after many years, hardly any young person has seen a penny. You go to your grandfather's attic to look for something, and by accident, spill several baskets of coins. Coins are flying around randomly. Your job is to catch the pennies with your basket as fast as possible. Some coins are falling into cracks in the wood. Some coins are not pennies (and you don't care about them). To complicate matters, a window washer pops by and he happens to be a numismatist (pronounced `noo-miz-ma-ticks`) someone who collects coins. He is also trying to catch the pennies, thus competing with you…

On Your Own

Come up with your own creative idea!

Required Checkpoints

1. Create 2 or more user controlled sprites; Create 1 or more randomly moving sprite.
2. When the random-sprite touches certain wall/object it changes movement and/or direction.

3. Final due date: When the user-controlled sprite and random-sprite touch, something happens to the score or counter. When the score or counter reaches are certain value, or condition, the animation or game is over.

Planning Worksheet

Part of the design process is planning. The “Pong Planning Worksheet” is an example of how to plan. Create a similar planning worksheet first before writing any code. Describe your own “game play” or animation rules using 1-2 paragraphs.

Grading Scheme/Rubric

Functional Correctness (Behavior)	
Players can control sprites with required keys	2 points
Random-sprite begins from a fixed location and returns there after certain event(s)	3 points
Random-sprite bounces correctly and moves within the window space	4 points
Random-sprite’s movement changes after certain event(s)	3 points
Score is changed when some interaction happens between sprites	3 points
Animation ends when some score is achieved	2 points
Winner or conclusion is announced when animation ends	1 point
Users can begin a new animation	1 point
Total	19 points
Technical Correctness (Implementation)	
Animation or game logic is smooth, polished, and intuitive	3 points
Program shows good creativity and effort	2 points
Program is well-documented and exhibits good style	2 points
Checkpoint 1	4 points
Checkpoint 2	4 points
Total	15 points
Grand Total	34 points

Unit 3

Lesson 3.1: Abstraction and Friends

Student Objectives

Students will be able to…

- Define the following terms in a computer science context:
  - abstraction
    - detail removal
    - generalization
    - procedural decomposition
- Describe how utilizing procedural decomposition can improve the readability and maintainability of algorithms and/or code
- Recognize opportunities to improve algorithms by abstracting or generalizing parts into subprocedures

Materials/Preparation

☐ Unit 3 Tips
☐ Student algorithms from lesson 0.2
☐ Paper/writing implements for each group of students

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introduce terminology, introduce first activity
10 minutes	Students rewrite PB&J algorithms
10 minutes	Review rewrites
10 minutes	Students write lasagna algorithms
10 minutes	Review new algorithms, debrief, wrap-up

Instructor's Notes

1. Introductory discussion

- Ask students to review (or recreate) their algorithms for preparing a peanut butter and jelly sandwich (from lesson 0.2)
- Point out that some parts of the algorithm are similar or redundant
- e.g. spreading peanut butter repeatedly until enough is on the bread; spreading peanut butter and jelly are basically the same operation
- Define and discuss the following terms:
- **abstraction:** removing the specifics that are not relevant in a given context
  - e.g. being able to drive a car without understanding how an internal combustion engine works
- **detail removal:** reducing the complexity of an algorithm or process by focusing on the important parts_
  - e.g. focusing simply on falling in the platform game instead of trying to create true parabolic motion
- **generalization:** combining a group of related concepts or processes into a single category
  - e.g. spreading any condiment or ingredient onto a slice of bread (butter, jam, peanut butter, mayo, etc.)
- **procedural decomposition:** breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts

2. Activity, part 1

- Instruct students to review their PB&J algorithms looking for opportunities for abstraction and procedural decomposition and rewrite the algorithms to improve readability
- Possible opportunities include: abstracting spreading peanut butter and jelly, looping when spreading to get enough of the ingredient, removing details with regard to opening packages, etc.
- Break students into groups and encourage students to make their new algorithms as concise as possible while still being easy to understand

3. Review

- Ask one or two groups to share their rewritten algorithms, pointing out the places where they abstracted or generalized
- If possible, choose groups with good approaches while circulating during the activity
- Ask other groups to point out any missed opportunities for or overuses of abstraction
- Discuss advantages to writing algorithms using abstraction
- More concise code, more structure, easier to follow, ability to ignore details until relevant

4. Activity, part 2

- Have students return to their groups and develop an algorithm for constructing and cooking a food that invloves layers (ex. lasagna).

- Encourage students to be aware of opportunities for abstraction and/or procedural decomposition when writing their algorithms
- As much as possible, students should abstract while writing the algorithm rather than writing a complete algorithm and then making edits
- Emphasize the balance between conciseness and readability

- Too much abstraction or generalization can make the algorithm hard to understand (e.g. end up with a "do stuff" procedure)

5. **Debrief**

- Ask at least two new groups to share their lasagna algorithms
- Ask students not in either group to compare and contrast the presented algorithms
- Point out, or ask students to point out, uses of, missed opportunities for, and overuses of abstraction
- Ask students how the skills learned in this lesson can be applied to programming
- Drive students to realize that code is simply an application of algorithms
- Tease the need for custom blocks (procedures)

BJC Lecture Suggestions

Background Information for Instructors

BJC Video Suggestion: BJC Lecture 1: Abstraction

- Abstraction: 11:40-15:40
- Generalization: 15:50-20:00

Accommodation/Differentiation

- A lasagna is used in this activity because it has many similar, repeated steps with different ingredients (layer of noodles, layer of sauce, layer of cheese, etc.). If students are not familiar with lasagna, another recipe with similar characteristics can be substituted.
- A club sandwich (or other "multi-decker" sandwich) can work, as can baking recipes (if the various dry ingredients that need to be mixed are treated separately)
- As in Lesson 0.2, students may go overboard with the decomposition in the second part of the activity. Encourage these students to think about the balance between brevity and usability, and remind them that the goal is not to write the *shortest* algorithm possible, but the *clearest* algorithm they can.

Forum discussion

Lesson 3.1: Abstraction and Friends (TEALS Discourse account required).

TEALS Unit 3 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 4, 8, 16

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Abstraction	Removing the specifics that are not relevant in a given context.
Detail Removal	Reducing the complexity of an algorithm or process by focusing on the important parts.
Generalization	Combining a group of related concepts or processes into a single category.
Procedural Decomposition	Breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts.
Custom Blocks	Allow for one to make their own programming blocks.
Command Block	puzzle-piece shaped block that executes a command (it causes an effect).
Reporter Block	Report a value, usually for use in another block's input slot.
Predicate Block	A hexagonal block that always returns a Boolean value (true or false).

Word	Definition
Argument	Any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block. The value that is “passed into” a parameter of a custom block
Say Block	The block gives its sprite a speech bubble with the specified text — the speech bubble stays until an another speech or thought block is activated, or the stop sign is pressed.

Lesson 3.2: Procedures

Student Objectives

Students will be able to...

- Build custom command blocks in SNAP
- Utilize detail removal and generalization to construct blocks that practice abstraction

Materials/Preparation

- ☐ Do Now 3.2: Drawing Squares
- ☐ Lab 3.2 handout - Drawing Shapes Again (Download in Word) (Link to PDF)
- ☐ Unit 3 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Lecture, demo, and introduce activity
20 minutes	Drawing Shapes activity
15 minutes	Review and wrap-up

Instructor’s Notes

1. Lecture and Demonstration

- Review with students the concepts of abstraction, generalization, and detail removal
- Provide an example of a poorly written algorithm that doesn’t demonstrate the concepts of abstraction, generalization, and detail removal and a well written algorithm that does demonstrate them.

- Ask students to point out the differences and ask what concepts they demonstrate that were previously learned
- Ask students to provide definitions of abstraction, generaralization, and detail removal refering to the sample algorithm provided
- Discuss when abstraction can be used to help simplify code
- Guide students to realization that much code is used in many similar circumstances and shouldn’t be rewritten
- Emphasize potential for errors when changes are made in addition to inconvenience when code is redundant
- Demonstrate constructing a custom command block
- Use a simple example, such as jumping (from the platform game)
- If needed, use part 2.1 (drawing a square) as a further example

2. Activity

- Students should complete the “Drawing Shapes (Again)” activity individually or in diverse pairs
- Work with students to be certain that they are using custom blocks and variables as described by the activity
- Point out places where code can be abstracted and generalized
- Emphasize conciseness and clarity in code

3. Review

- Discuss one or two student submissions
- Point out differences between different students' solutions
- Point out missed opportunities for abstraction (if any)
- Discuss how this custom block can be useful

BJC Lecture Suggestion

BJC Video Suggestion: BJC Lecture 4: Functions MIT Scratch –> BYOB Snap (Development of SNAP, Demo) 10:00-11:30 BYOB-Custom Blocks (Explains functions with examples) 11:30-14:50

Accommodation/Differentiation

- In addition to attempting the bonus in the lab, advanced students can be encouraged to write a new custom block that draws a specified number of the given shape (for example, 2 squares or 5 hexagons). The number of shapes should be taken as user input.
- Struggling students can be given solution code for all of section 1. Work with these students to answer the questions in part 1.5 so that they are prepared to attempt the custom block authoring.

Forum discussion

Lesson 3.2: Procedures (TEALS Discourse account required).

TEALS Unit 3 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 4, 8, 16

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Abstraction	Removing the specifics that are not relevant in a given context.
Detail Removal	Reducing the complexity of an algorithm or process by focusing on the important parts.
Generalization	Combining a group of related concepts or processes into a single category.
Procedural Decomposition	Breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts.
Custom Blocks	Allow for one to make their own programming blocks.
Command Block	puzzle-piece shaped block that executes a command (it causes an effect).
Reporter Block	Report a value, usually for use in another block's input slot.
Predicate Block	A hexagonal block that always returns a Boolean value (true or false).
Argument	Any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block. The value that is "passed into" a parameter of a custom block
Say Block	The block gives its sprite a speech bubble with the specified text — the speech bubble stays until an another speech or thought block is activated, or the stop sign is pressed.

Do Now 3.2 Drawing Squares

Go to this starter project

Fill in the scripts to draw the same size square 3 different ways. Make sure you follow the directions in the comments so you're using the right blocks for each script. Press space key to clear the stage in between each test of your scripts.

Lab 3.2 - Drawing Shapes (Again)

In this lab, you will write code to draw regular polygons. But this time, you will write custom blocks and abstraction to write more efficient code.

Simple Shapes

1. Write a SNAP script (or find one you've already written) to draw a square.
2. Write a SNAP script (or find one you've already written) to draw an equilateral triangle.
3. Write a SNAP script (or find one you've already written) to draw a regular pentagon.
4. Write a SNAP script (or find one you've already written) to draw a regular octagon.
5. Look over the four programs from above. Do you notice sections that are very similar? What sections might be able to be abstracted into a separate block?

If You've Seen One

1. Take your script from above that draws a square and turn it into a custom block.
2. Modify your custom block to use a variable for the number of sides instead of the number 4. Set that variable's value to be 4 so that you still draw a square.
3. Now, see if you can modify your block so that it will work correctly for any number of sides greater than 2. Look closely at the angles in your four scripts section 1 and see if you can spot a pattern.
4. Use your custom block in a program that asks the user for a number and then draws a regular polygon with that many sides. If the number given is less than 3, give an error message and do not draw anything.

Bonus: Sizing Things Up

1. Modify your custom block and program from the previous section so that the user can specify both the number of sides and the size of each side. Be sure to utilize generalization and detail removal to make your program and block as clear and concise as possible.

Grading Scheme/Rubric

Lab 3.2 Criteria	
1 Custom block draws square	0.25 points
2 Use variable for number of sides	0.5 points
3 Works for all sides > 2	0.5 points
4a Program asks for sides, calls block	0.5 points
4b Program handles input <= 2	0.5 points
Bonus Program and block handle size of sides	0.5 points
Code clear and concise	0.25 points
PROJECT TOTAL	**3.0 points**

Lesson 3.3: Customization I

Learning Objectives

Students will be able to...

- Build custom SNAP blocks that take arguments

Materials/Preparation

- ☐ Do Now 3.3: Jumping
- ☐ Lab 3.3 handout (Let Me Check My Calendar) (Download in Word) (Link to PDF)
- ☐ Unit 3 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and introduce activity
20 minutes	Custom block argument activity
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Lecture

1. Introduce block arguments
   - Define **arguments:** An argument is any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block.
   - Ask students to speculate on risks of relying on variables instead of arguments
   - Variables could be changed by code other than the custom block, variable names could be changed causing errors, etc.
   - Emphasize importance of custom blocks being self-contained
   - Custom blocks should continue to function correctly independent of any other changes in the program
   - Custom blocks should work correctly anywhere in the program and not require specific setup or cleanup
   - Point out how frustrating code would be if blocks like Move 10 steps block required setting a variable with a specific name to work
2. Demonstrate declaration and usage of arguments
   - Point out that arguments are very similar to script variables, except their values come outside the block
   - Explain argument types
   - Only discuss text, numbers, and Booleans
   - Other types can be mentioned, but won't be used in the course
   - Ask students to describe why restricting types is important
   - Point out that arguments are passed by value
   - Specifically, changing the value of an argument inside a custom block will typically NOT change the value at the call site
   - Arguments example
   - Basic argument in pen category
   - Pass by value example in "Variables" category

2. Activity

- Students should complete the "Let Me Check My Calendar" activity individually or in pairs
- This lab consists of a series of independent custom blocks. The blocks need not necessarily be completed in the order given, but are roughly in order of difficulty.
- The bonus (part 3.4) requires implementing a fairly complex formula.

3. Debrief

- Ask a different student to provide their solution to each part.

- If time allows, discuss multiple solutions to each part.
- Emphasize differences and encourage discussion about advantages and disadvantages.
- Point out corner cases and cases where typed arguments are particularly helpful.
- For example, avoiding try to find out whether "bubblegum" is a leap year

Accommodations/Differentiation

- Part 3.4 is a fairly complex formula and should be a challenge for advanced students.
- Struggling students should focus on section 1. The problems in section 2 represent more complex algorithms, but not necessarily any more difficult usage of arguments.

Forum discussion

Lesson 3.3: Customization I (TEALS Discourse account required).

TEALS Unit 3 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 4, 8, 16

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Abstraction	Removing the specifics that are not relevant in a given context.
Detail Removal	Reducing the complexity of an algorithm or process by focusing on the important parts.
Generalization	Combining a group of related concepts or processes into a single category.
Procedural Decomposition	Breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts.
Custom Blocks	Allow for one to make their own programming blocks.
Command Block	puzzle-piece shaped block that executes a command (it causes an effect).
Reporter Block	Report a value, usually for use in another block's input slot.
Predicate Block	A hexagonal block that always returns a Boolean value (true or false).
Argument	Any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block. The value that is "passed into" a parameter of a custom block
Say Block	The block gives its sprite a speech bubble with the specified text — the speech bubble stays until an another speech or thought block is activated, or the stop sign is pressed.

Do Now 3.3 Jumping

Follow link to a project with a dog that can move left and right in response to arrow keys. Make the following modifications to the code:

1. Program a walking animation for the dog. Hint: it has 2 costumes.
2. Add the code below to the forever block to make the dog "jump". What is the problem with this jump code when you test it? How should jump work when the player presses the spacebar?

   Key Press jump
3. Program gravity by making a custom Motion block "gravity" and adding it inside the forever loop. In the "gravity" custom block use an if statement to implement gravity. Hint: If not touching ground or platform

color, go down by a small amount.

4. Remember to save your work.

Lab 3.3 - Let Me Check My Calendar

In this lab, you will write some custom blocks that take arguments and are useful for calculations involving dates and calendars.

Basics

1. Write a custom SNAP block called “month name” that takes a number between 1 and 12 as an argument and says the name of the corresponding month.
2. Write a custom SNAP block called “day name” that takes a number between 1 and 7 as an argument and says the name of the corresponding day. For our purposes, the week begins on Sunday.
3. Write a custom SNAP block called “days in” that takes a month name as an argument and says how many days are in that month. Assume a non-leap year.

Going Farther

1. Write a custom SNAP block called “is a leap year” that takes a year number as an argument and says whether or not that year is a leap year.
    - A year is a leap year if the year is a multiple of 4 that is not a multiple of 100 (e.g. 1984), or if it is a multiple of 400 (e.g. 2000). Years that are multiples of 100 but not multiples of 400 are NOT leap years (e.g. 1800). See Wikipedia for more detail.
2. Write a custom SNAP block called “is a valid date” that takes a month name and a date as arguments and says whether or not that date exists in that month. For example, the 31st is a valid date in January, but not in June. The 5th is a valid date in every month, and the 40th is not a valid date in any month.
3. Write a custom SNAP block called “day in year” that takes a year number and a number between 1 and 366 and says the date that corresponds to that numbered day of the specified year. For example, in non-leap years day #1 is January 1, day #32 is February 1, day #365 is December 31, and day #185 is July 4. Give an error message if the number is 366 and a non-leap year is specified.
4. BONUS: Determine the day you were born. Write a custom SNAP block called “day of week” that takes a month name, date, and year as arguments and says the day of week on which that date falls in that year. See http://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week for information on finding the day of the week from a date.

Grading Scheme/Rubric

Lab 3.3 Criteria	
1.1 month name	0.5 points
1.2 day name	0.5 points
1.3 days in	0.5 points
2.1 is leap year	0.5 points
2.2 is a valid date	0.5 points
2.3 day in year	0.5 points
2.4 Bonus: day in week	0.5 points
PROJECT TOTAL	**3.0 points**

Lesson 3.4: Customization II

Learning Objectives

Students will be able to...

- Build custom reporter and predicate blocks in SNAP

Materials/Preparation

- ☐ Do Now 3.4: Practice using Arguments
- ☐ Lab 3.4 handout (If My Calculations Are Correct...) (Download in Word) (Link to PDF)
- ☐ Unit 3 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Lecture and introduce activity
25 minutes	Custom reporter activity
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Lecture

1. Introduce reporter blocks
   - Ask students to find blocks with the reporter shape (round) and speculate as to what they do
   - Point out familiar examples, such as x position block, answer block, pickrandom block, etc.
   - Explain the concept of reporting (returning) a value, and how reporter blocks are used to provide values to commands
   - Emphasize that reporters do not (and should not) perform any action—they are used to compute values which are in turn used by commands
2. Introduce predicate blocks as a special case of reporter blocks
   - Emphasize that predicates must return true or false
   - Be aware, but don't necessarily tell students, that SNAP does not enforce this
   - Point out examples, such as Touching Block, less than block, and block
   - Ask students why it might be useful to differentiate predicates from other reporters
   - Only predicates can be used in a conditional

2. Activity

- Students should complete the "If My Calculations Are Correct..." activity individually.
- This lab consists of a series of independent custom blocks. The blocks need not necessarily be completed in the order given.
- Work with students to ensure they are testing their blocks properly and reporting correct values.
- Part 2.2 is challenging and requires traversing a String.

3. Debrief

- Ask a different student to provide their solution to each part. If time permits, go over multiple students' work for each part.
- Point out differences and discuss advantages and disadvantages to different approaches.
- Emphasize that custom blocks do not have to be long and complicated to be useful.

Accommodations/Differentiation

- Struggling students should focus on just the first 2 or 3 parts of the lab. Even if they cannot move on to the more difficult problems, getting used to defining custom reporters is helpful and important.
- Advanced students who finish quickly can be utilized to assist other students.
- Students who struggle in math may not be familiar with the distance formula used in part 2.1. Help students to translate the math into SNAP code, but understanding the formula and its applications is not necessary for the activity.
- If most students are not equipped to handle this math, a simpler computation, such as area of a triangle or average of three numbers, can be substituted.

Forum discussion

Lesson 3.4: Customization II (TEALS Discourse account required).

TEALS Unit 3 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 4, 8, 16

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Abstraction	Removing the specifics that are not relevant in a given context.
Detail Removal	Reducing the complexity of an algorithm or process by focusing on the important parts.
Generalization	Combining a group of related concepts or processes into a single category.
Procedural Decomposition	Breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts.
Custom Blocks	Allow for one to make their own programming blocks.
Command Block	puzzle-piece shaped block that executes a command (it causes an effect).
Reporter Block	Report a value, usually for use in another block's input slot.
Predicate Block	A hexagonal block that always returns a Boolean value (true or false).
Argument	Any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block. The value that is "passed into" a parameter of a custom block
Say Block	The block gives its sprite a speech bubble with the specified text — the speech bubble stays until an another speech or thought block is activated, or the stop sign is pressed.

Do Now 3.4 Practice using Arguments

Yesterday you created a "gravity" custom block where a sprite would fall to the ground and stop when it touched a color.

However, the sprite always falls at the same rate. How would we get the sprite to fall at different rates?

We will solve this problem by adding an argument to the custom block named "gravity". We can then pass the rate we want the sprite to move.

1. Open yesterday's do now with the "gravity" custom block.
2. Add an argument named "rate" to the "gravity" block by clicking the "+" next to the custom block name.
3. Change the "move" block to move "rate" spaces by dragging the rate variable to the "move" block.

4. You can now pass different rates in your “gravity” custom block call located in the sprite’s forever loop.

Lab 3.4 - If My Calculations Are Correct

In this lab, you will write custom reporter blocks to perform a number of useful calculations and computations.

Simple Computations

1. Write a custom SNAP reporter block called “min” that determines which of two numbers is smaller and reports that value. If the two numbers are equal, report either one.
2. Write a custom SNAP reporter block called “max” that determines which of two numbers is larger and reports that value. If the two numbers are equal, report either one.
3. Write a custom SNAP predicate block called “between” that determines if a number is between two other numbers. If the first number is equal to either of the other two numbers or is between them, the block should report “true”.
4. Write a custom SNAP predicate called “at least as long as” that determines whether or not word has at least a specified number of letters.

Stepping Things Up

1. Write a custom SNAP reporter block called “distance to” that computes and reports the distance from a sprite’s position to another point. Use the “x position” and “y position” blocks to determine the sprite’s position. Remember that the formula for the distance between points (x1, y1) and (x2, y2) is sqrt((y2-y1)$^{2+(x2-x1)}$2).
2. Write a custom SNAP predicate block called “contains letter” that determines whether or not a word contains a particular letter. You can consider upper-case and lower-case letters to be different for the purposes of matching.

Grading Scheme/Rubric

Lab 3.4 Criteria	
1.1 min	0.25 points
1.2 max	0.25 points
1.3 between	0.5 points
1.4 at least as long as	0.5 points
2.1 distance to	0.5 points
2.2 contains letter	0.5 points
PROJECT TOTAL	**2.5 points**

Unit 3 Quiz Customization

Learning Objectives

Formative assessment on student progress: To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Materials/Preparation

- TEALS Classes can access the Quizzes by logging into the TEALS Dashboard and navigating to “Additional Curriculum Materials” -> “Intro CS Curriculum”.

- You will need to log in using incognito mode on the browser.
- See Additional Curriculum Resources for further instructions.

Lesson 3.5: Platform Game Projec

Learning Objectives

Students will be able to…

- Use loops, variables, and Boolean expressions to implement a Super Mario Bros. style platform game.
- Practice good debugging skills to correct issues as they arise while programming.

Materials/Preparation

- Reference to the Platform Game Project: Platform Game (Download in Word) (Link to PDF)
- (Optional) Advanced Version of the project: Advanced Platform Game (Download in Word) (Link to PDF)
- (Optional) Printouts of the specification
- A few suggested themes available for students who may struggle to come up with their own.
- Suggestions for characters, power-ups, goals, etc.
- Unit 3 Tips

Pacing Guide

Duration	Description
Day 1	
5 minutes	Welcome, attendance, bell work, announcements
30 minutes	Review unit topics
20 minutes	Introduce project
Days 2-9	
5 minutes	Welcome, attendance, bell work, announcements
10-15 minutes	Review
30-35 minutes	Lab time
5 minutes	Exit ticket

Instructor's Notes

1. Review/Introduction

- Play a review game (such as GrudgeBall) to remind students of the skills and concepts have been learned in this unit.
  - Basic loops
  - Nested loops
  - User input
  - Conditional blocks
  - Advanced (conditional) loops
  - Boolean logic
- Remind students that their solutions to previous assignments are an excellent resource when trying to accomplish similar tasks.

2. Introduce project

- Walk students through the project specification, pointing out important details, potential pitfalls, and requirements.

- If students are unfamiliar with the concept of a platform game, spend a couple minutes demonstrating one for the class. If you have a SNAP or Scratch version, that works best, but an online game (such as Super Mario Flash) will work as well.
- Discourage students from simply recreating a game they are already familiar with (and using copyrighted assets) and encourage them to be creative and design their own characters and world instead.
- Static screens are somewhat easier to implement than smooth scrolling, but scrolling is doable with some scaffolding.

3. Project

- This project is a summative assessment for the unit. Students should be demonstrating mastery of all the skills covered.
- Most students will require roughly 6-10 hours of total work time to complete the project
- Assess the progress of your students regularly using such techniques as asking them to demonstate their incomplete programs, tracking questions asked during lab time, and/or utilizing peer reviews.
- Adjust the amount of time allowed for the project to fit the needs of your students
- It is vital that nearly all students complete the project before moving on
- If most students have the ability to work on SNAP assignments at home, the amount of in-class time provided can be reduced if necessary.
- If this approach is taken, be sure to make accommodations for students who are *not* able to work at home, such as after school lab hours
- Ensure that students are able to ask questions in class throughout the project
- See the standard Lab Day Lesson for detailed plans for lab days.

Accommodations/Differentiation

- If any students do not have the ability to work at home, ensure enough in-class time is provided to complete the assignment, offering extensions if necessary.
- Advanced students can be encouraged to extend their games beyond the minimum required number of screens, add more features (power-ups, "boss" enemies, secret screens, etc.), create multiple "levels" that must be worked through, or any other extension.
- Struggling students can be exempted from certain requirements or given starter code.
- If scaffolding this project, provide code to perform some of the various actions required in the game (changing screens, moving the character, defeating enemies, etc.) and ask the students to fill in the triggers and conditions for using these routines.

Forum discussion

Lesson 3.5: Platform Game Project (TEALS Discourse account required).

TEALS Unit 3 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 4, 8, 16

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Abstraction	Removing the specifics that are not relevant in a given context.
Detail Removal	Reducing the complexity of an algorithm or process by focusing on the important parts.
Generalization	Combining a group of related concepts or processes into a single category.
Procedural Decomposition	Breaking a problem down into smaller subtasks, usually to increase readability and/or maintainability, often by applying one of the above concepts.

Word	Definition
Custom Blocks	Allow for one to make their own programming blocks.
Command Block	puzzle-piece shaped block that executes a command (it causes an effect).
Reporter Block	Report a value, usually for use in another block's input slot.
Predicate Block	A hexagonal block that always returns a Boolean value (true or false).
Argument	Any area in a block that accepts user input, or another block. It could be a Boolean Block or a value placed inside of a variable or block. The value that is "passed into" a parameter of a custom block
Say Block	The block gives its sprite a speech bubble with the specified text — the speech bubble stays until an another speech or thought block is activated, or the stop sign is pressed.

Project 3: Advanced Platform Game

Students will implement a side*scrolling platform game (Example: Super Mario Bros.) in Snap!.

Overview

Platform games are among the most widely recognized types of video games. Composing about one third of all console games at the peak of their popularity, platform games are characterized by their relative simplicity and by the common gameplay element of jumping across suspended platforms (hence the name) to avoid falling into a hazard. Platform games also typically include enemy characters, items that grant the hero special abilities ("power*ups"), and a"checkpoint" system that allows the player to restart from partway through a game or level when he or she dies.

Big Ideas * Personal design interests require the evaluation and refinement of skills; Tools and technologies can be adapted for specific purposes

Consider using the familiar platform game concept to not only entertain, but also educate or bring awareness to social concerns, an environmental issue, or some other context.

Example 1: In Pacific Northwest, there are many efforts exist to promote and guide a sustainable future of wild Pacific Salmon and their habitat. Students can create a game where the player can help salmon swim back to the stream where they were born, jumping upstream through ladders and fishways, battling hazards and enemies (predators, fishermen). Game play can introduce elements of education and awareness.

Example 2: The West Coast has been battling with more Wildfires than ever in history. Students can create a game where a traveler drives through different towns, parks, and highways in your area, extinguishing hazards (cigarette butts, campfires), battling enemies (fires out of control, irresponsible tourist), enforcing fire bans, reporting fires, being a good neighbor, implementing home prevention, rescuing animals, and so on. Game play can introduce elements of geography (parks and forests, lakes and waters) and climate and terrain (temperature, animal habitats). Read more about wildfires in here:

How can the classic platform game be adapted to a specific purpose, cause, or context that interests you, or impacts your community?

Behavior

Gameplay

In a platform game, the player controls a hero who moves throughout the world attempting to reach an endpoint and/or accomplish a goal. Along the way, the hero encounters hazards such as pits (into which he or she can fall) and enemies (which can either move or be stationary). The hero has a finite number of chances (known as "lives") to achieve his or her objective. Each time the hero succumbs to a hazard, a life is lost and the player must try again.

The Hero The hero moves around the world under the player's control. The hero can perform three basic actions: move left and right (controlled by the arrow keys) and jump (by pressing the space bar). As the hero moves throughout the world, he or she is subject to gravity. This means that when the hero jumps or moves off the edge of a platform, he or she should return quickly to the ground (or another platform). The hero should never fall through a platform or the ground.

Screens Your platform game will be a side*scrolling game. In this style of game, the scenery changes as the player moves horizontally across the screen. While many modern side*scrollers (including Super Mario Bros.) scroll smoothly as the player moves, our game will use a simpler system and consist of screens. Each screen represents one section of the overall world in which the game takes place. The hero should be able to move freely around each screen, but when the hero reaches the far right edge of a screen, the next screen should appear and the hero should be placed on the far left edge at the same height. Your game must include three distinct screens, and each screen must have one hazard.

Hazards and Lives Your program should include one of each of the following types of hazards:

- A falling hazard (a hole, pit, or other opening) into which the hero can fall if he or she does not jump to avoid it. Falling to the bottom of this hazard causes the hero to die.
- A stationary enemy that does not move, but that causes the hero to die if it is touched.
- A moving enemy that also causes the hero to die if it is touched, but that moves in some way. Note that an enemy can be either a character (like Goombas in Super Mario Bros.) or an environmental hazard (such as spikes). When the hero dies by either falling down a falling hazard or touching an enemy, he or she loses a life. If the hero has lives remaining, one should be lost, used power*ups and defeated enemies should be reinstated, and the hero should be placed back at the left edge of the current screen. Otherwise, the game is over and a suitable message should be displayed.

PowerUps While moving through the world, the hero may obtain power*ups that grant him or her special abilities. Examples can include increased jumping power, invulnerability, or the ability to destroy enemies. These abilities should be temporary, and there should be some visual indication when the hero has access to them. A power*up should appear as an item (sprite) in the world. The hero will obtain the abilities by touching the power*up sprite, at which point the power*up should disappear. Your game should include two distinct power*ups, one of which is required for the hero to win the game. In addition, one of your power*ups should be hidden, meaning that the player must take some action before he or she can obtain its abilities. For example, in Super Mario Bros., Mario must jump into a special block to make many power*ups appear. A hidden power*up should not be visible until it is triggered by the hero, and the hero should not be able to obtain the abilities until the power*up is revealed.

Winning the Game There should be some clear end goal that, when achieved by the hero, ends the game in victory. This victory condition should be obvious even to a new player. You need not (and probably should not) provide written instructions to the player about this condition—use easily identifiable visual indicators such as flags, doors, etc.

Reset Button At any point during gameplay, if the player presses the 'z' key, the game should reset to its initial state. The game state after pressing the 'z' key should be indistinguishable from when the game first begins. This means, among other things, that:

- the hero should return to the left edge of the first screen, lose all special abilities, and be given back all his or her lives
- any destroyed enemies should be reinstated and replaced in their original positions
- obtained power*ups should become available and revealed power*ups should be hidden

Implementation Details

Design and Creativity

Your program should be well*designed and have a unifying theme, characteristic, or style. This can be a particular style of artwork, common colors, and/or related types of characters. In addition, you should show some effort and creativity in your design. Do not simply recreate an existing game or use only ideas put forth in this spec. Come up with some original concepts for characters, backgrounds, powerups, etc. and utilize them in your game. If you make us say “Wow!” it may even be worth some extra credit. Using copyrighted assets (including characters or artwork from an existing game), even with modification, is not allowed.

Custom Blocks

Throughout your program, you should use custom blocks to generalize common operations and increase the readability and maintainability of your code. Your program must include at least three custom blocks, at least one of which must take arguments. Do not limit yourself to just three blocks: use custom blocks (including arguments and reporters) anywhere you feel it will help your code. Part of your grade will be based on not only meeting the minimum usage requirement but also on your decisions of when, where, and how to use custom blocks.

Documentation

In addition to functioning well, your program must be well documented and readable. This includes, but is not limited to, things such as:

- organizing your scripts so that they can be read and comprehended easily
- giving your sprites meaningful names
- naming and using your variables well
- including comments to describe the structure of your program and any particularly complex or unintuitive pieces of code

Required Snap! Elements

Your program must include, at a minimum, the following Snap! code elements:

- At least two variables
- At least one conditional (if or if-else) statement
- At least two messages, one of which must be received and responded to by multiple sprites

Peer Feedback

As part of the software development experience on this project, you will participate in a peer review with one or more of your classmates. Near the end of the project, you will play another student’s game and provide him or her with notes and comments. Your partner(s) will also play your game and offer the same feedback. Describe to your partner what parts of the game you liked. You should offer suggestions for features that could be improved or changed as well as look for bugs in the program you are reviewing. Keep your comments constructive and professional! Don’t just point out things you don’t like—explain your thinking and propose solutions. Also, restrict your comments to things that can be reasonably addressed. Do not tell your partner that he or she made a poor choice of theme and should start over, for example. After receiving your peer feedback, you should consider the comments carefully and respond. You will be expected to turn in the feedback provided to you and identify ways in which you modified your game in response to the feedback you received.

Curriculum Competencies * Design Sharing As you create software, you will need to put yourself in the end*user’s position, and imagine yourself using the software. At the same time, providing objective, critical analysis and feedback of other people’s work will not only benefit others, but also help you in your own design work. Being able to welcome, and provide, feedback are essential skills for a software developer.

Required Checkpoints

1. Screens should be designed; the hero should be able to move and jump; gravity should work; reset button should be functional
2. Hazards and enemies should be present; death should work properly.
3. Lives, power*ups, and victory should be implemented; all other required program components must work

Grading Scheme/Rubric

Functional Correctness (Behavior)	Value
Left and right arrows and space bar control hero's movement, and hero does not move through walls	2 points
Hero is subject to gravity, and does not fall through platforms	2 points
Game consists of three screens	1 point
Game contains: One falling hazard (pit) One stationary enemy One moving enemy	3 points
Player loses a life when falling down the falling hazard or touching an enemy, and hero restarts on current screen after death.	2 points
Player starts with three lives, and game ends when player is out of lives	2 points
Game contains two powerups, one of which is hidden and one of which is necessary to win the game	3 points
Hero has a clear goal to win the game	1 point
Gameplay is clear and intuitive, even to a brand new player	1 point
Game resets when the 'z' key is pressed	2 points
Technical Correctness (Implementation)	Value
Program is well designed visually and has a consistent theme	2 point
Program shows good creativity and effort	2 points
Program is well*documented and exhibits good style	2 points
Program includes at least two variables	2 points
Program includes at least two messages, at least one of which is received and reacted to by multiple sprites	2 points
Program includes at least one conditional statement	1 point
Program includes at least three custom blocks, including at least one with arguments	4 points
Custom blocks, including arguments and reporters, are used where appropriate	2 points
Provide valuable playtest feedback to at least two other students	2 points
Obtain and respond to playtest feedback from at least two other students	2 points
Checkpoint 1	2 points
Checkpoint 2	4 points
Total	27 points
Grand Total	46 points

Project 3: Platform Game

Adapted from Forest Ridge High School's Mario Project.

Students will implement a side-scrolling platform game (For Example Super Mario Bros.) in Snap!.

Overview

Platform games are among the most widely recognized types of video games. Composing about one third of all console games at the peak of their popularity, platform games are characterized by their relative simplicity and by the common gameplay element of jumping across suspended platforms (hence the name) to avoid falling into a hazard. Platform games also typically include enemy characters, items that grant the hero special abilities ("power-ups"), and a "checkpoint" system that allows the player to restart from partway through a game or level when he or she dies.

Details

Controllable Sprite

Your program should have a “Mario” sprite. It does not necessarily need to look like Mario (a simple stick figure will suffice). Mario should respond to user input. Specifically:

- Mario should clearly “face” the right when you push the right arrow key
- Mario should clearly “face” the left when you push the left arrow key
- Mario should perform an animated “in-place” walk when you hold left or right arrow key
- Mario should jump based on some input (you can decide the key or mouse click)

Moving Scenery Sprites

You should have scenery sprites that move based upon Mario’s traveling on the level. It is up to you to decide the level scenery, but you should meet the following requirements:

- There should be at least two scenery sprites (Example: A mountain and a tree)
- You should layer these sprites relative to Mario and each other. For example, Mario should always be “in front” of any background scenery sprites
- Scenery sprites move relative to Mario as he moves. For example, when you hold down the right arrow key, the background sprites should move from right to left in the stage.
- Following with the layers, scenery sprites should move at different speeds so that one seems farther away. For example, a faraway mountain should move more slowly than a nearby tree.
- Scenery sprites should roll over/re-appear when they hit the edge of the stage. For example, when Mario is walking to the right, the scenery Sprites should re-appear on the right side of the stage when they roll off the left.

On-Ground Enemy

There should be an on-ground enemy for Mario to contend with. Specific criteria here include:

- There is at least one on-ground enemy
- The enemy sprite moves towards Mario, independent of whether Mario is moving (e.g. regardless of whether the user is pressing an arrow key).
- The enemy sprite reappears/rolls-over when he hits the edge of the stage
- The enemy sprite is animated when he is moving towards Mario
- If Mario does not jump he runs into the enemy and the game ends with an appropriate message <HINT: you can use the “STOP ALL” block to end all scripts
- It should be possible for Mario to jump over the enemy.

Programming Habits

We will again look for you to incorporate good programming habits in your code:

- Using Start and Stop blocks.
- Making sure you initialize state appropriately so that your program is repeatable.
- Add comments to your code so it is easy to understand.
- Always keep in mind all Good Programming Skills you’ve been taught

Additional Extensions

Once you complete the above, you can extend your program. Some suggestions:

- Include flying enemies for Mario to dodge or duck
- Keep score based on how many objects Mario gets by [Hint: Use a variable and show it on the screen]

- Have Mario jump to 'grab' an object that offers Mario extra points or more powerful abilities (such as jumping higher or not being killed when he runs into an enemy). The objects must appear at random times and move smoothly as Mario runs

We will demo the most interesting projects in class, so be creative!

Grading Criteria

The detailed list for what we will use to grade your projects is below. Please review your projects before submitting to be sure you meet all of them. If you have any questions on whether you meet a requirement, please ask us!!

Requirement	Value
Mario turns to the right on right arrow key	5
Mario turns to the left on left arrow key	5
Mario performs an in-place animated walk if you hold down either arrow key	5
Mario jumps based on some input	5
There are least two scenery sprites	5
Scenery sprites are layered with Mario (e.g. appropriate layering blocks are in your program)	5
Scenery sprites move based on Mario's movement	5
Scenery sprites move at different speeds (e.g. far away versus near)	5
Scenery sprites roll over when the fall of the stage	5
There is at least one on-ground enemy	5
Enemy sprite always moves towards Mario	5
Enemy sprite re-appears/rolls over correctly	5
Enemy sprite is animated when it moves	5
If Mario does not jump, he runs into the enemy and the game ends nicely and properly	10
Mario can jump over the enemy	5
Good programming #1: Program has clear start and stop	5
Good programming #2: Program is repeatable and initializes state correctly	5
Good programming #3: Use of comments in your code	5
Good programming #4: General skills you've been taught	5
Good programming #5: Custom blocks used to break down program into logical parts	5
Extra Credit: Include flying enemies for Mario to dodge or duck	5
Extra Credit: Keep score based on how many objects Mario gets by	5
Extra Credit: Have Mario jump to 'grab' an object that offers Mario extra points or more powerful abilities. The objects must appear at random times and move smoothly as Mario runs	10
Extra Credit: Create Your Own Extension	0-10
TOTAL POINTS	**105** (135 Possible with all the extra credit)

Unit 4

Lesson 4.1: Intro to Lists

Learning Objectives

Students will be able to...

- Explain the concept of a "list" in a programming context

- Identify scenarios in which lists are useful

Materials/Preparation

- ☐ Unit 4 Tips
- ☐ Paper/writing implements for each group of students
- ☐ Large poster paper and markers will allow for display of the algorithms, but standard paper will work fine

- ☐ Video Explanation of Lists

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introductory discussion
10 minutes	Students write Solar System algorithm
10 minutes	Debrief
10 minutes	Translate algorithms into SNAP pseudocode
10 minutes	Debrief and wrap-up

Instructor's Notes

Introductory discussion

- Shortly describe the Solar System - Sun, planets, etc, and hook your students in with a way (in programming) to collect and store new information and discoveries.

Activity

- In small groups, students will write an algorithm for adding new Solar System discoveries to a collection of older ones. As with the PB&J activity, the process should be complete and detailed so that a person can unambiguously follow the steps.
- The process itself will seem relatively simple. Ensure students think not only about the steps to be taken but the necessary materials and resources.
- Ensure that the algorithm would work for any discovery of any name size and does not make assumptions. e.g SUPER MEGA COOL ASTEROID, etc.
- Pay particular attention to when the various materials are needed.
- The key here is that the student will need to constantly be referencing this same pool of discoveries e.g. Pool, Pool + 1 discovery, Pool + 1 discovery + another etc. etc.
- Once discussion of the algorithm is complete, instruct students to think about how they would write a SNAP program to complete this task. They need not write actual code, but should write pseudocode to attempt to solve the problem.

- After groups have finished, explain to them that usually scientists organize their discoveries alphabetically not by recently discovered. Challenge them to create an algorithm that organizes their new discoveries with their old discoveries alphabetically. After this is done, challenge them to create it in SNAP.
- The ultimate conclusion should be that they need a way to store the entire collection of discoveries, and operate on individual parts of the collection.

Debrief

- In attempting to write pseudocode, students should realize that they need variables to store the discoveries, but do not know ahead of time how many variables will be necessary.
- Clever students may want to simply store the message in a single variable using the "join" block. This approach can work if they choose an unambiguous delimiter (space won't work if there are multi-word sections of the message), and is effectively the same as using a list.
- Point out that, thus far, they have not seen a way to store an arbitrary number of data values—they have needed a separate variable for each value, which must be created ahead of time.
- Briefly introduce the concept of a list as a means of storing multiple values in a single location. Lists in SNAP have the useful property of not having a static size, so any number of values can be added at any time.

Accommodations/Differentiation

- As in the PB&J activity, discourage stronger students from dominating the conversation and instead ask them to take on a leadership role and help other group members find issues.
- Struggling students can be given permission to use a higher level of abstraction, ignoring certain details that other students will attend to.

Forum discussion

Lesson 4.1: Intro to Lists (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Lesson 4.2: Static Lists

Learning Objectives

Students will be able to...

- Create static lists in SNAP
- Access elements of a list
- Add and remove elements from a list

Materials/Preparation

- ☐ Do Now 4.2: Letters of a Word
- ☐ Lab 4.2 handout (You Talkin' to Me?) (Download in Word) (Link to PDF)
- ☐ [Snap! List Components] (Download in Word), (Download PDF)
- ☐ Unit 4 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and introduce activity
25 minutes	Grammar Activity
10 minutes	Debrief and wrap-up

Instructor's Notes

Lecture

- Review the concept of a list from the previous lesson
- Ask students to brainstorm examples of when lists could be useful
- To store an unknown number of values (e.g. a bunch of student test scores, shopping list, the songs of your favorite music artist)
- To store a collection of related values as one entity (e.g. the number of absent students each day over a week, how often a video on YouTube in a week)
- Demonstrate creating lists in SNAP
- Use the Snap! Lists Components file to demostrate the Snap! list structure
- Use the variadic (taking a variable number of arguments) "list" block to create a simple list
- Point out the format in which lists are displayed (gray box with red elements)
- Show that lists can be assigned to variables like other values
- Emphasize that the list is considered a single value, even though it consists of multiple values

- Point out and explain basic list operations blocks
- The "item," "add," and "delete" blocks will be most important. The "length" block will be useful as well.
- Point out that these blocks all take a list as an argument.

Activity

- Students should complete the You Talkin' to Me? activity individually or in pairs
- Encourage students to be creative with their word lists
- Don't allow students to fixate on the exact grammatical correctness of generated phrases and sentences
- If this is a major concern, choose words for the lists such that generated phrases will always be grammatically correct

Debrief

- Ask a student to present and discuss their solution to each step
- Emphasizes uses of lists and encourage students to discuss and think about why lists were necessary
- Ask students to consider if the tasks would have been doable without lists

Accommodations/Differentiation

- In addition to the bonuses in the lab, advanced students can attempt further extensions of the grammar, including conjunctions, non-transitive verb phrases, and/or recursive rules (e.g. multiple adjectives).

- A more complex context-free grammar for English sentences can be found here
- Struggling students should focus on generating a noun phrase from only a few words. The other parts of speech and phrase types can be omitted without losing the key learning objectives.
- Non-native English speakers or those with low literacy may struggle with the grammatical concepts here. Since the grammar is not the key objective, feel free to scaffold liberally and/or substitute a different type of grammar.
- Other grammar examples, including arithmetic expressions (which are a good simple substitute) can be found here.
- You can also provide a grammar supplement with example sentences (example from (Woodward English)

Forum discussion

Lesson 4.2: Static Lists (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

Word	Definition
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Do Now 4.2 Letters of a Word

Go to this starter project and read and run the script. Once you understand how it works, modify the script so that the sprite says every letter in the word.

Sentence Generator

In this lab, you will create a simple sentence generator using lists.

Let's Talk

1. Create a variable for each part of speech below, and set each variable to hold a list of words that fit that part of speech. Some examples are given, but feel free to use your own.

Part of Speech	Example words
noun	giraffe, monkey, boy, girl, elephant, …
verb	jumps, runs, sleeps, sits, dances, …
adjective	big, small, loud, silly, young, old, sleepy, …
adverb	quickly, excitedly, angrily, happily, sadly, …
article	a, the
preposition	under, over, around, near, beside, …

Reporting Phrases

1. Write a custom reporter block called "noun phrase" that reports a noun phrase where each word is chosen randomly from the lists you created. A noun phrase consists of an article, an adjective, and a noun in that order.
2. Write custom reporter blocks like "noun phrase" for each of the phrase types listed below.

Phrase type	Construction
prepositional phrase	preposition, noun phrase
verb phrase	adverb, verb, preposition phrase
sentence	noun phrase, verb phrase

Making Sentences

1. Write code so that when you press the space bar, a random sentence is generated and a sprite says the resulting sentence.
2. BONUS: Modify your code so that a noun phrase can either be the construction from part 1.2 or a single proper noun (e.g. a person's name). Your code should randomly decide which version of a noun phrase to use.
3. BONUS: Modify your code so that a verb phrase can sometimes leave out the prepositional phrase. Your code should randomly decide to include the prepositional phrase or not.

Changing Our Vocabulary

1. Write a script so that when the 'n' key is pressed, the user is prompted for a new noun and that noun is added to list of nouns. After that point, the new noun entered by the user should be able to appear in sentences.
2. Write scripts like the one you wrote in part 4.2 to add words to the other lists. Use the keys listed below.

Key	Part of speech
v	verb
j	adjective
d	adverb
a	article
p	preposition

3. Write a script so that when the 'x' key is pressed, the user is asked for one of the parts of speech and then for a number (n). Your script should remove the nth item from the list of words for the specified part of speech. For example, if the user types in "verb" and "3" then you should remove the 3rd word from the list of verbs. The removed word should no longer appear in sentences.

Grading Scheme/Rubric

Lab 4.2 Criteria	
1.1 6 lists	0.5 points
2.1 "noun phrase" reporters	0.5 points
2.2 3 phrase and sentence reporters	0.5 points
3.1 sentence script	0.5 points
3.2 Bonus: Add proper nouns	0.25 points
3.3 Bonus: Make prepositional phrase for verb phrases	0.25 points
4.1 Add nouns	0.25 points
4.2 Add other 5 parts of speech	0.25 points
4.3 Remove a part of speech	0.5 points
PROJECT TOTAL	**3.5 points**

Lesson 4.3: List Practice I

Learning Objectives

Students will be able to...

- Traverse a list, accessing each element one at a time
- Perform operations combining all elements in a list
- Select defined subsets of elements in a list

Materials/Preparation

- ☐ Do Now 4.3: Quote of the Day
- ☐ Lab 4.3 handout (Guess Who) (Download in Word) (Link to PDF)
- ☐ Unit 4 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and demonstration

Duration	Description
30 minutes	Guess Who Activity
5 minutes	Wrap-up

Instructor's Notes

1. Lecture

1. In small diverse groups, ask students to consider how to count the number of students with July birthdays in the room
   - Provide only a few minutes to work through this– students need not write full, formal algorithms, but simply desribe an approach
   - Have groups share with each other and work together to come up with a single approach, hopefully settling on asking each student if s/he has a July birthday and counting the number of yeses (or something similar).
2. Define **traversal** as the process of accessing each element of a list in order and performing some operation.
   - Call out that the operation can be anything, and may not actually be performed on every element
   - Provide a few examples of possible operations (say each person's name, count the number of females, add up the total number of siblings, find the average GPA, etc.)
   - Explain that traversing is how many problems involving lists are solved.
3. As a group, develop sample code for a simple list traversal, such as the following:

   Simple List Traveral

   - Ask leading questions to help students write each line of code, one a time, then gradually put the pieces together ("build up" approach)
   - Point out that the "say" block can be replaced by any code (including larger blocks of code for more complex operations), but that the rest of the script will typically be the same
   - Emphasize that the "index" variable is keeping track of where we currently are in the list, and can be used in the traversal operation if wanted, as in:

   Use index In Loop Example

2. Activity

- Students should complete the Guess Who activity individually or in pairs
- Students will be performing several traversal operations, some of which simulate mapping, filtering, or reducing/folding the list. You can discuss these operations if you feel the class can handle it.
- Part 1.3 requires use of the "join" block—be sure that the students are comfortable using this block.

3. Wrap-up

- Ask students to briefly describe how the various parts of the lab were similar or different
- Hopefully the students find that the scripts were quite similar for each part.
- Ask students to describe something they found challenging about the lab
- You will go over the correct solutions to the lab in the next lesson

BJC Lecture Suggestion

BJC Video Suggestion :BJC Lecture 9: Recursion

- Movie "Inception" as an example of recursion 0:00-0:50
- Recursion 0:50-1:40
- Recursion Demo in Snap 1:40-17:00
- Overview 17:00-21:00
- Definition of Recursion 21:00-24:30

- Examples of Recursion (You Already Know It!) 24:30-26:20
- Trust the Recursion 26:22-29:40
- Summary of Recursion 29:40-End

Accommodations/Differentiation

- Advanced students can attempt more complex filters in part 2.3 (such as finding all names that contain at least two vowels) and/or more advanced maps in part 1.3 (such as greeting each person by first initial).
- Struggling students should focus on parts 1.2, 2.1, and select items from part 2.3. Partial code can be provided.

Forum discussion

Lesson 4.3: List Practice I (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Do Now 4.3 Quote of the Day

1. Create a list with 10 inspirational quotes. You may use the internet for ideas.
2. Use the following block to pick a random quote of the day:

   item

Lab 4.3 - Guess Who

In this lab, you will create a list of names and then look through the list pulling out different subsets of the names.

Role Call

1. Create a list of names with at least six different names (e.g. the names of your favorite music artists, family members, authors, celebrities, etc.). Try to vary the names as much as you can.

2. Write a SNAP script to welcome each person to the program by name, one at a time. (For example, “Welcome, John.” “Welcome, Kayla.” “Welcome, Michael.”) Make sure not to modify the list of names when you run the script—you’ll want the list again later. Also make sure your script still works even if the list of names changes.
3. Write a new SNAP script that welcomes all the players at once. So, for example instead of saying “Welcome, John,” “Welcome, Kayla,” and “Welcome, Michael.” separately, you’re new script should say “Welcome John, Kayla, and Michael.” Start by writing a script that can say all the names on one line, then try to add the commas and “and”. Make sure your script works correctly no matter how many names are in the list.
4. BONUS: Modify your code so that instead of using a pre-determined list of names, the user can enter the names to be included in the list one at a time. You’ll need to decide how to determine when the user has entered all the names.

I’m Looking For

1. Write a script that says every other name in a list one at a time when the space bar is pressed. Use the same list of names from above. For example, if the list is [Eric, Sally, Michelle, John, Sam, Caleb], the names Eric, Michelle, and Sam would be said.
2. Write a script that says the names in the list one at a time in reverse order when the ‘0’ key is pressed. For example, if the list is [Eric, Sally, Michelle, John, Sam, Caleb], the names Caleb, Sam, John, Michelle, Sally, and Eric would be said.
3. Write scripts so that when each of the following keys is pressed, the corresponding subset of names from the list is said one at a time.

When this key is pressed…	Say the names in the list that…	For example…
1	Have more than four letters	Sally, Michelle
2	Start with the letter ‘c’	Caleb
3	End with the letter ‘y’	Sally
4	Are not the first two or last two names in the list	Michelle, John
5 (optional)	Contain the letter ‘e’	Eric, Michelle, Caleb

Grading Scheme/Rubric

Lab 4.3 Criteria	
1.2 Welcome by name	0.25 points
1.3 Welcome all at once	0.25 points
1.4 Bonus: Enter list, then welcome all at once	0.25 points
2.1 Every other name	0.25 points
2.2 Reverse order	0.5 points
2.3.1 > 4 letters	0.25 points
2.3.2 start with C	0.5 points
2.3.3 End with y	0.5 points
2.3.4 Not first 2 or last 2	0.5 points
2.3.5 Bonus: Contain e or E	0.25 points
PROJECT TOTAL	**3.5 points**

Lesson 4.4: List Practice II

Learning Objectives

Students will be able to…

- Traverse a list, accessing each element one at a time

- Perform operations combining all elements in a list
- Select defined subsets of elements in a list

Materials/Preparation

- ☐ Do Now 4.4: Traversing List
- ☐ Lab 4.4 handout (Number Cruncher) (Download in Word) (Link to PDF)
- ☐ Unit 4 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review and debrief lab 4.3
25 minutes	Activity
15 minutes	Debrief and wrap-up

Instructor's Notes

Review

- Ask students to define "traversal" and outline the basic code pattern
- You need not write actual code here, but have students mention the key points (index variable, use index to access each item, repeat length of list, etc.)
- Review solutions to lab 4.3
- Ask a student to provide their solution to each part
- Discuss errors or flaws in each solution
- Point out similarities between each part, emphasizing that the basic code pattern is the same each time with only the operation performed on each item changing
- If you feel students can handle it, you can further classify various traversals (maps, filters, folds)
- If necessary, provide the basic code pattern for a traversal again:

  simple list traversal

Activity

- Students should complete the Number Cruncher lab individually
- The operations in this lab are fairly similar to those in lab 4.3, but work with lists of numbers instead of names. As a result, students should progress more quickly.
- As in lab 4.3, help students realize that the basic code pattern in each part will be the same.

Debrief

- Ask one or two students to share their solution to each part of the lab
- If all students seem to grasp the concept, not all parts need to be reviewed
- If skipping some parts, be sure to review at least parts 1.1 (a fold), 2.1 (a map) and 2.2 (a filter)
- Point out that the solutions from labs 4.3 and 4.4 will look quite similar, even though in lab 4.3 the lists contained names in in lab 4.4 they contained numbers
- The traversal code pattern is the same regardless of the type of elements in the list

Accommodations/Differentiation

- Advanced students should complete both bonus parts of the lab (1.5 and 2.3) and then assist struggling students.

- Students who need more assistance should focus on parts 1.3, 2.1, and 2.2. Ensure that all students are able to complete at least these three parts before concluding the lesson.

Forum discussion

Lesson 4.4: List Practice II (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Do Now 4.4 Traversing List

1. Describe what it means to "traverse a list".
2. List 2 different blocks you can use to traverse list and how you would use them.

Number Cruncher

In this lab, you will continue practicing processing lists, this time using lists of numbers instead of words.

Summarizing Numbers

1. Write a custom SNAP reporter block called "sum" that takes a list as an argument and reports the sum of all the numbers in the list. You can assume that all items in the list passed as the argument will be numbers, though you should not assume anything else.
2. Write a custom SNAP reporter block called "average" that takes a list as an argument and reports the average of all the numbers in the list. As above, you can assume that all items in the list passed as the argument will be numbers, but you should not assume anything else.
3. Write a custom SNAP predicate block called "includes negative" that takes a list as an argument and reports true if the list contains at least one negative number, and false if all numbers are non-negative.
4. Write a custom SNAP predicate block called "increasing?" that takes a list of numbers as an argument and reports true if each value in the list is greater than or equal to the one before it.

5. Write a custom SNAP reporter block called “maximum” that takes a list as an argument and reports the largest number in the list.

Transforming Lists

1. Write a custom SNAP reporter block called “make all positive” that takes a list of numbers as an argument and reports a new list that is the same as the argument, except all negative numbers have been replaced by their absolute value.
2. Write a custom SNAP reporter block called “only evens” that takes a list of integers as an argument and reports a new list that contains only the even numbers from the argument list. The result list should have its values in the same order as the original list, but with the odd integers removed. (Remember that “mod” block can be useful in determining whether or not a number is even.)
3. BONUS: Write a custom SNAP reporter block called “add all” that takes two list of numbers as arguments and returns a new list that contains the sum of the corresponding values in each argument list. For example, if the arguments to “add all” are (1, 4, 6) and (2, 2, 3), the result should be (3, 6, 9). You can assume the two lists will be the same size.

Grading Scheme/Rubric

Lab 4.4 Criteria	
1.1 sum reporter	0.25 points
1.2 average reporter	0.25 points
1.3 includes negative predicate	0.25 points
1.4 increasing? predicate	0.25 points
1.5 maximum reporter	0.25 points
2.1 make all positive reporter	0.5 points
2.2 only evens reporter	0.25 points
2.3 Bonus: add all reporter	0.25 points
PROJECT TOTAL	**2.25 points**

Lesson 4.5: Sequential Search

Learning Objectives

Students will be able to...

- Explain the sequential search algorithm
- Implement several variations of sequential search

Materials/Preparation

- Do Now 4.5: List Tracing
- Lab 4.5 handout (It’s Around Here Somewhere) (Download in Word) (Link to PDF)
- Unit 4 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Lecture and demonstration
25 minutes	It’s Around Here Somewhere Activity
10 minutes	Debrief and wrap-up

Instructor's Notes

Lecture

- Ask students to consider how to determine if a particular person is in the room or not
- At first, you will likely get answers like "call out the person's name" or "look around." Press the students to come up with a method that will always work, including when the person is not present. If necessary, ask them to pretend they are a computer.
- Point out that solutions like "look around" are too high-level, and in reality, there is a lot more going on (such as looking at each person individually).
- Guide students to the process of checking if each person is the one they are seeking, in some deterministic order, until they have either found the person or checked everyone. Emphasize that the absence of the person is only confirmed when everyone has been checked, but that the presence is known as soon as the person is found.
- Get students to recognize that this process is a traversal of the people in the room.
- Explain that the process of traversing a list looking for a particular item is known as a "sequential search."
- Ask students to think about the efficiency of this algorithm. Emphasize best, worst, and average cases (both what those cases are and how long they take).
- You need not get into specific runtimes or big-O notation, but students should have a basic understanding of the fact that the speed of the search is dependent on the size of the list.
- If students seem prepared, ask them to speculate under what circumstances you might be able to do better (eventually leading to binary search).
- Show the code for a basic sequential search: basic sequential search
- Point out that this code is another variation of a traversal.
- Emphasize that this is only one variant of sequential search. The specifics of what to report can vary (true/false, index in list, item itself, etc.).

Activity

- Students should complete the It's Around Here Somewhere activity individually or in pairs.
- Each part asks students to write a slightly different sequential search. In all cases, make sure students are clear on what they should be reporting, both when the desired item is found and when it is not.

Debrief

- Ask one or two students to provide their solutions to each part
- Point out the similarities in each solution, emphasizing that the algorithm remains constant and only the value that is reported changes.

Accommodations/Differentiation

- Advanced students can be encouraged to explore more efficient searching algorithms (specifically binary search).
- Struggling students should focus on just parts 1.1 and/or 2.1 of the lab. Remind these students that they are starting with a traversal and simply changing what happens to each item.

Forum discussion

Lesson 4.5: Sequential Search (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Do Now 4.5 List Tracing

For each script below, describe what the sprite does when that script is triggered. Number your answers. If you have time, check your answers by assembling these scripts in SNAP.

You will need to add the "Words, sentences" library to your project to use the "list->sentence" block. In Snap:

1. Click the File Icon in the upper left corner next to the "Snap!" image.

2. Click "Libraries…"
3. Select "Words, sentences"
4. Click "Import" The "list->sentence" block will be one of the blocks added to "Operators"

   List Do Now

It's Around Here Somewhere

In this lab, you will implement several custom blocks performing variants of sequential search.

You There

1. Write your own version of the SNAP "contains" block, which takes a list and a value as arguments and reports true if the value is anywhere in the list and reports false otherwise. You should NOT use the existing "contains" block in your implementation.

Where

1. Write a custom block called "index of" that takes a list and a value as arguments and reports the index at which the value is found in the list, if it is there. If the value is not present anywhere in the list, report -1. So, for example, if the list is (2, 3, 5, 7, 11) and the value is 5, "index of" should report 3. If the list is (2, 3, 5, 7, 11) and the value is 4, "index of" should report -1.

Tell Me More

1. Write a custom block called "first e-word" that takes a list as an argument and reports the first word in the list that starts with the letter 'e'. If no such word exists, report a blank (nothing).

2. BONUS: Write a custom block called "first word that starts with" that takes a list and a letter as arguments, and reports the first word in the list that starts with the given letter. If no such word exists, report a blank (nothing).

Grading Scheme/Rubric

Lab 4.5 Criteria	
1.1 contains block	0.5 points
2.1 index of block	0.5 points
3.1 first e-word block	1.0 points
3.2 BONUS: first word that starts with block	0.5 points
PROJECT TOTAL	**2.5 points**

Unit 4 Quiz Lists

Learning Objectives

Formative assessment on student progress: To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Materials/Preparation

- TEALS Classes can access the Quizzes by logging into the TEALS Dashboard and navigating to "Additional Curriculum Materials" -> "Intro CS Curriculum".
- You will need to log in using incognito mode on the browser.
- See Additional Curriculum Resources for further instructions.

Lesson 4.6: Guess My Word Project

Learning Objectives

Students will be able to…

- Use lists to implement a complete version of "Guess My Word"
- Exercise good programming practices to produce code that is not only functional but also elegant and well-written

Materials/Preparation

- Do Now 4.6: Gentle Guess My Word
- Project 4 - Guess My Word (Download in Word) (Link to PDF)
- Guess My Word planning worksheet: (Download in Word) (Link to PDF)
- Link to an online version of a word guessing game, such as http://www.searchamateur.com/Play-Free-Online-Games/Bubbletoonia-Cake-Deal.htm
- A word list from which words can be selected for the game, such as Word List
- Unit 4 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Review and introduce project

Duration	Description
As needed	Lab time for Guess My Word project

Instructor's Notes

- Review/Introduction
- Review the various skills and concepts have been learned so far in the unit. Consider making it a Kahoot! game.
- Put particular emphasis on the maintenance of lists and traversals
- Remind students that their solutions to previous assignments are an excellent resource when trying to accomplish similar tasks.
- Walk students through the project specification, pointing out important details, potential pitfalls, and requirements.
- Focus students' attention on the checkpoints to help them avoid becoming overwhelmed.
- Help students import the word list into a SNAP list, either by right-clicking on the list view and selecting "import" or by parsing the online list directly using the `http://` block. This is not one of the objectives of the project, so feel free to provide starter code if you prefer.
- Emphasize that generating the "word pattern" or "blanks" and keeping that sequence up-to-date with each guess is the hardest part of the assignment, and should not be overlooked.
- Remind students that it will be important to keep straight what each variable and list in the program is used for. Each variable or list should have a single purpose, and those purposes should never be conflated.
- Project
- This is a summative assessment project. Students should be given at least a few days in class to work on the project. The exact schedule should be determined by your teaching team based on overall class capability and other factors.
- If most students have the ability to work at home, you can consider reducing the amount of in-class time provided and requiring students to spend time working at home.
- Provide a means for students to ask questions throughout the project and provide assistance as needed.

Accommodations/Differentiation

- If any students do not have the ability to work at home, ensure enough in-class time is provided to complete the assignment, offering extensions if necessary.
- Advanced students can be encouraged to add extensions such as:
  - Enabling guessing of the entire word
  - Allowing the user to specify the length of the word to be guessed
  - Keeping statistics (win-loss record, fewest guesses, etc.) across games
  - Struggling students can be given starter code or exempted from certain features
  - Though it is the most difficult part of the assignment, tracking the "blanks" is the best practice working with lists and should be preserved.

  - The requirements to display a graphical figure, track repeated guesses, and specifically recognize a win can be removed for students in need of significant simplification.

Forum discussion

Lesson 4.6: Guess My Word Project (TEALS Discourse account required).

TEALS Unit 4 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 19

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Programming List	Called an 'array' in most programming languages, this holds lots of values.
Lists Block	A block which controls a list.
Operator Block	A block that performs math functions and string handling.
List Element	A part of a list.
Traverse	Go through or travel across an item/list.
Transform	Create a new list based on the values of the old list.
Sequential Search Algorithm	A method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.
Index Variable	Keeps track of where you currently are in a list.
Join Block	Links two values together and reports the result.
Contains Block	Checks an operator block for a particular variable.

Do Now 4.6 Guess My Animal

Go to this site to play this word guessing game.

1. How could you use lists from this unit to help implement a version of this game in Snap!?
2. What other creative ideas can you come up with to visualize the number of chances a player has to guess the letters? (come up with at least 2)

Project 4: Guess My Word

Students will implement a Snap! "Guess my Word" word game.

Overview

In Guess My Word, one player (the "chooser") chooses a secret word and another player (the "guesser") attempts to guess the word one letter at a time. If a guessed letter appears in the word, all instances of it are revealed. If not, the guesser loses a chance. If the guesser figures out the secret word before he or she runs out of chances, he or she wins. If not, the player who chose the word wins. Chances are tracked using a figure drawing of your choice. You can build a bear or build an ice cream cone, or disassemble a snowman, be creative. The figure is drawn one part at a time, and the guesser loses when the entire figure has been drawn or disappears. This game is also the basis for the TV game show Wheel of Fortune.

Behavior

Gameplay In this implementation of Guess My Word, the computer will take on the role of the "chooser" and the human player will be the "guesser." The computer will secretly choose a word from a list (see below) and show the player how many letters are in the word by displaying a sequence of blanks (underscores). Then, the computer will begin asking for guesses. If the player guesses a letter that is in the secret word, all blanks representing an instance of that letter should be replaced by the letter. If the guessed letter is not in the word at all, the player should lose a chance and a new part of the figure being built should appear or disappear if figure is being disassembled. If the player guesses a letter he or she has already guessed, he or she should not lose a chance, even if that letter is not in the word. If the player guesses all letters in the word, he or she wins. If the figure being built is completed, the player loses. In either case, the secret word should be revealed after the game is over.

Sprites Your game will need to include at least three sprites: the figure being built or disassembled, a "host" sprite that asks the player for a guess and informs him or her whether the guess is correct, and an "assistant" sprite that tells the player the status of the secret word. You may use more sprites if you think they are appropriate. The host and assistant should have clear roles and should never do each other's job.

Word Status As the game is played, the player should be shown the current guessed status of the secret word. Letters that have been correctly guessed should be shown in the correct locations. Unguessed letters will appear as blanks. At the beginning of the game, no letters will have been guessed, and the only information shown to the player will be a sequence of blanks, with one blank for each letter in the secret word. As the player guesses letters correctly, blanks representing guessed letters should be replaced by those letters. So, for example, if the secret word is "screwdriver" and the player has guessed `e`, `s`, `r`, and `d`, the current word status would be: `s _ r e _ d r _ _ e r`.

Chances The player will have six "chances" to guess the word. Guessing a correct letter does not cost a chance. Each missed chance will cause a new piece of the figure being built to appear or disappear if figure is being disassembled. The six pieces will depend on the figure you choose. Example 1, a snowman can have the following parts: head, eyes, left arm, right arm, middle snowball, and bottom snowball. Example 2, a bear can have the following parts: head, body, left arm, right arm, left leg, and right leg. Example 3, an ice cream cone can have the following parts: cone, 2 scoops of ice cream, fudge, sprinkles, and a cherry. If you would like to be more creative with the appearance, feel free to do so. No matter what your figure looks like, though, it should include these six pieces and no more.

Game End The game can end in one of two ways:

- If the player has guessed the complete secret word, he or she wins.
- Otherwise, if the player has run out of chances and the complete figure has been drawn, or disassembled as in the Snowman sample solution the player loses.

In either case, when the game ends the host should stop asking for guesses. The host should inform the player whether he or she won or lost, and the assistant should reveal the entire secret word.

Implementation Details

Word List/Secret Word You will be provided with a list of words from which the secret word should be chosen for each game. Instructions for importing the word list:

- To import the list, set a variable "imported words" to be a list.
- Download and save as a text file on your computer.
- Click the checkbox on the "imported words" variable so that it is visible on the stage.
- Right click on the stage image to find the 'import...' command. Select the correct file to import.
- Set another variable to be a list of words by splitting the imported words by line.

  Importing List Instructions

Documentation In addition to functioning well, your program must be well-documented and readable. This includes, but is not limited to, things such as:

- organizing your scripts so that they can be read and comprehended easily
- giving your sprites meaningful names
- naming and using your variables, lists, and custom blocks well
- including comments to describe the structure of your program and any particularly complex or unintuitive pieces of code

Required Snap! Elements Your program must include, at a minimum, the following Snap! code elements:

- At least two lists, once of which must be used to track guessed letters
- Custom blocks as appropriate, including arguments and reporters

Required Checkpoints

1. Be able to select a secret word, keep track of which letters have been guessed, determine if each letter guessed is in the secret word or not
2. Be able to announce the current status of the word, showing letters that have been guessed and blanks for other letters.
3. Be able to play a full game of Guess My Word, identify correct and incorrect letters, display the figure, and inform the player whether they have won or lost.

Grading Scheme/Rubric

Functional Correctness (Behavior)	
Computer randomly chooses a secret word	1 point
Host repeatedly asks for a letter and announces whether that letter is in the secret word	2 points
Assistant displays the correct secret word status after each guess	4 points
Player loses a chance and a piece of the figure appears when a guess is incorrect	3 points
Host informs player when he or she guesses a letter that has already been guessed; player does not lose a chance	2 points
Game ends with player victory if the entire secret word is guessed	2 point
Game ends with player defeat if the player runs out of chances	2 point
Secret word is revealed when game ends	1 points
SubTotal	**17 points**
Technical Correctness (Implementation)	
Program is well-designed visually and has a consistent theme	2 point
Program is well-documented and exhibits good style	2 points
Program shows good creativity and effort	3 points
Program includes at least two lists	2 points
Program uses custom blocks with arguments and reporters appropriately	2 points
Program tracks guessed letters using a list	2 points
Obtain and respond to playtest feedback from a parent or guardian	2 points
Checkpoint 1 (4/30)	4 points
Checkpoint 2 (4/30)	4 points
SubTotal	*19 points*
Total	**40 points**

Unit 5

Lesson 5.1: Intro to Cloning

Learning Objectives

Students will be able to...

- Explain why prototyping and clones can be useful
- Describe how complex goals can be accomplished using cloning

Materials/Preparation

- ☐ Unit 5 Tips
- ☐ Lab 5.1 Handout (Connect the Dots) (Download in Word) (Link to PDF)
- ☐ If possible, create large poster-sized versions of the dot grids (each grid is 12 x 3).

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introduce activity
25 minutes	Activity
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Introduce activity

- Inform students that they will be drawing some figures by following specific instructions
- Emphasize that students must follow all instructions in the lab carefully
- Throughout the activity, ask students to think about other ways they could accomplish the same goals and the advantages and disadvantages of each approach.

2. Activity

- Split students into diverse groups of at least six. If the number of students is not an exact multiple of six, create a few groups of seven and have students take turns being "active."
- Students should follow the steps in the lab, being careful to act as a group.
- In each part, the group will draw the letter 'C' six times, using slightly different instructions.
- Students should, hopefully, notice that in part 3, they are able to achieve similar but not exactly the same results by all following the same instructions. (Though each student draws a 'C', they are not all in the same location.) In each part, they were able to improve the efficiency and clarity of the instructions.

3. Debrief

- Ask each group to share their answers to the questions at the end of each part.
- Discuss how this approach could be applied to coding.
- Introduce the terms **prototyping** and **cloning** as (mostly) synonyms:
- **prototyping:** creating a single "master" entity that defines the behavior for a group of objects, then creating many copies of the prototype to duplicate the behavior

BJC Lecture Suggestions

Fibonacci and Fibonacci Series Video 7:45-11:45

Background Information for Instructors

BJC Lecture 11:Recursion II Alijia Yan

- Mobile World Congress 0:00-2:15
- Recursion:Factorials (Factorial (n)+ n! 2:30-7:40
- Fibonacci and Fibonacci Series Video 7:45-11:45
- Fibonacci Ex: fin(n) Math and SNAP blocks 11:50-13:15
- Example of Recursion: Counting Change 13:20-17:30
- Call Tree for "Counting Change" with SNAP example 17:35-22:50
- Summary of Recursion 25:40-26:21

Accommodations/Differentiation

Forum discussion

Lesson 5.1: Intro to Cloning (TEALS Discourse account required).

TEALS Unit 5 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 5, 9, 12, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Cloning	A feature that allows a sprite to create a clone, or semi-duplicate, of itself, while the project is running. Clones of a sprite will be the same as the original or parent sprite but as a separate instance. Clones inherit the parent's scripts, costumes, sounds, and properties, yet they can then be modified.
Global Variable	A variable that can be used by all of your sprites.

Lab 5.1 - Connect the Dots

In this lab, you will follow sets of instructions to investigate how clones work.

Part 1 - All Together Now

Each of the six students in your group should follow these instructions. Each student should then follow his or her instructions, completing each corresponding step together. (For example, everybody should complete their step ii. at the same time.) Everyone should work on the same grid.

1. Put your pen on the first dot in the bottom row of the grid below.
2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen. Dots

What was the result of the group following the instructions? Were there any difficulties you encountered? Does it seem like the result was intended?

Part 2 - To Each His Own

1. Give each member of the group one of the sets of instructions below. Each student should then follow his or her instructions, completing each corresponding step together. (For example, everybody should complete their step ii. at the same time.) Everyone should work on the same grid.

Person #1

1. Put your pen on the first dot in the bottom row of the grid below.
2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen.

Person #2

1. Put your pen on the third dot in the bottom row of the grid below.

2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen.

Person #3

1. Put your pen on the fifth dot in the bottom row of the grid below.
2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen.

Person #4

1. Put your pen on the seventh dot in the bottom row of the grid below.
2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen.

Person #5

1. Put your pen on the ninth dot in the bottom row of the grid below.
2. Draw a line to the dot to the right.
3. Draw a line to the dot to the left.
4. Draw a line to the dot above.
5. Draw a line to the dot above.
6. Draw a line to the dot to the right.
7. Lift your pen.

Person #6

1. Put your pen on the eleventh dot in the bottom row of the grid below.

2. Draw a line to the dot to the right.

3. Draw a line to the dot to the left.

4. Draw a line to the dot above.

5. Draw a line to the dot above.

6. Draw a line to the dot to the right.

7. Lift your pen.

   Dots

What was the result of following the instructions? Does this seem like the most efficient way to achieve this goal? What would happen if the shape needed to be changed? What problems could that cause?

What might be a better or more efficient way to accomplish the goal of these instructions?

Part 3 - The Same But Different

Assign each member of your group a different number from 1 to 6. Each of the six students in your group should then follow these instructions. Each student should then follow his or her instructions, completing each corresponding step together. (For example, everybody should complete their step ii. at the same time.) Everyone should work on the same grid.

1. If you are person #1, put your pen on the first dot in the bottom row of the grid below.
2. If you are person #2, put your pen on the third dot in the bottom row of the grid below.
3. If you are person #3, put your pen on the fifth dot in the bottom row of the grid below.
4. If you are person #4, put your pen on the seventh dot in the bottom row of the grid below.
5. If you are person #5, put your pen on the ninth dot in the bottom row of the grid below.
6. If you are person #6, put your pen on the eleventh dot in the bottom row of the grid below.
7. Draw a line to the dot to the right.
8. Draw a line to the dot to the left.
9. Draw a line to the dot above.
10. Draw a line to the dot above.
11. Draw a line to the dot to the right.
12. Lift your pen.

Dots

Was the result of these instructions any different than in part 2? Do you think this process was more or less effective than the process from part 2? Why?

Lesson 5.2: Cloning Sprites

Learning Objectives

Students will be able to...

- Demonstrate the difference between sprite and global variables
- Explain how cloning and prototyping simplify working with numerous similar sprites in the same program
- Create prototype sprites and clones of the prototype sprite
- Explain the difference between a "master" sprite and a "clone" sprite

Materials/Preparation

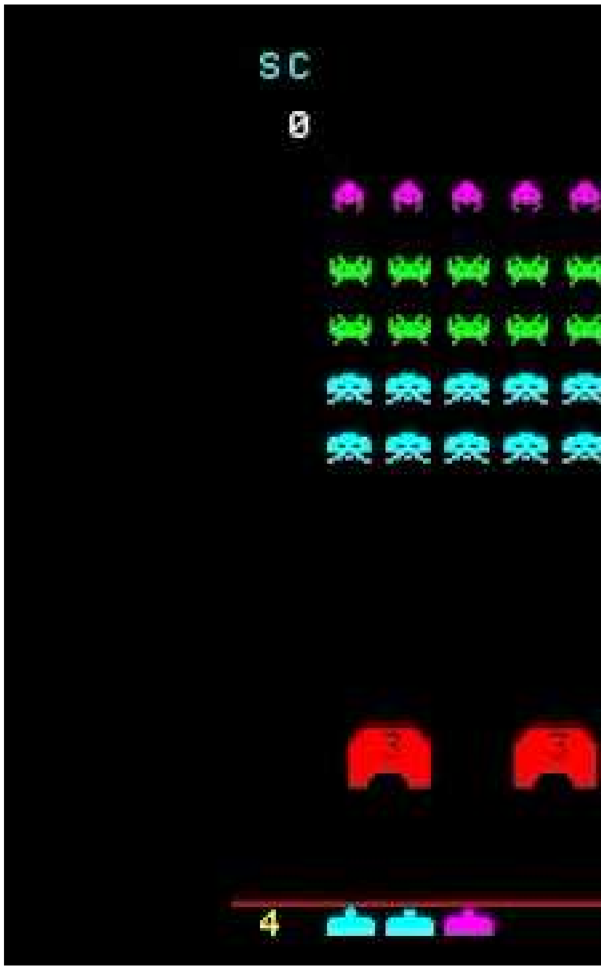
- ☐ Do Now 5.2: Star Wars Troopers
- ☐ Lab 5.2 handout (Lots of Balls) (Download in Word) (Link to PDF)
- ☐ Test out the lab on student machines before class (cloning in SNAP! can bring the web browser to a crawl on some machines)
- ☐ Unit 5 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Introductory discussion
30 minutes	Lots of Balls lab
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Introductory discussion



Example Game Show students a brief video demonstrating space invaders

Discussion

- Ask students to think about how we create all the "invaders"?
  - Answer with current knowledge: make one invader sprite, and copy it many times while adjusting the copies as necessary
- Ask what might go wrong or be bad about this approach
  - If anything in an invader changes, it will need to be changed many times
  - Lots of sprites clogging up the program that are all basically doing the same thing

Demonstration

- Introduce cloning as the automated way of doing the manual copying
- Demonstrate how to create a clone using Create a Clone of Block
- Point out that clones inherit all aspects of the "master" or "prototype" sprite, including scripts
- Emphasize the importance of using When I start as a clone Block to ensure clones don't duplicate out of control

2. Activity

- Students should complete the Lots of Balls lab
- This lab will largely duplicate the code shown in the lecture part of the lesson– that's OK
- Students should focus on ensuring they are differentiating between "master" sprites and "clone" sprites, and that the stage is serving as the main "driver" for the program

Accommodations/Differentiation

- Advanced students can attempt to assign properties to clones (color, size, etc.) so that not all clones look alike. To do this, they will need to use a global variable to temporary hold the value that can be "claimed" by the clone.
- Struggling students should focus on just creating a single clone from the prototype and not worry about creating multiple clones.

Forum discussion

Lesson 5.2: Cloning Sprites (TEALS Discourse account required).

TEALS Unit 5 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 5, 9, 12, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Cloning	A feature that allows a sprite to create a clone, or semi-duplicate, of itself, while the project is running. Clones of a sprite will be the same as the original or parent sprite but as a separate instance. Clones inherit the parent's scripts, costumes, sounds, and properties, yet they can then be modified.
Global Variable	A variable that can be used by all of your sprites.

Do Now 5.2 Star Wars Troopers

1. In Star Wars, what are "clone troopers"? Do an internet search to find out and write a sentence to answer the question.
2. Go to this SNAP project. Duplicate the sprite (and scripts) so that there are 5 clone troopers that move from the right side of the screen to the left side in a straight line when green flag is clicked. The starting and ending positions of the sprites are shown below.

Start:

clone formation

End:

clone formation end

Lab 5.2 - Lots of Balls

In this lab, you will use cloning to create many identical sprites without having to reprogram each one individually.

Part 1 - Follow the bouncing sprites

1. Create a SNAP program that contains a single sprite, choose from available costumes (i.e. basketballs, hearts, stars, balloons, Alonzo, doves). When the green flag is clicked, the sprite should go to the center of the stage, pick a random direction, and start moving in the chosen direction. If the sprite hits a wall, it should bounce off and keep moving.

2. Modify the program to be controlled by the stage rather than by the sprite itself. When the spacebar is pressed, the stage should broadcast a message that triggers the sprite's movement. Pressing the spacebar again should restart the sprite's movement, including a new speed and new direction.
3. What would you need to do to add a second bouncing sprite (that behaved in the same way) to the program? What about 10 sprites? 100 sprites? What would happen if you wanted to change the speed of all the bouncing sprites in the program after you had created 100?

Part 2 - Clones

1. Modify your program so that, instead of a single sprite restarting each time the spacebar is pressed, a new clone of that sprite is created. You'll want to use the Create a CLone of and When I starts as a clone blocks in place of Broadcast block and When I Receive block.
2. What happens to the original ("master") sprite each time the spacebar is pressed? Does that seem useful? What role should the original sprite play now that we're cloning?
3. Modify the program so that the original ("master") sprite hides at the beginning of the program and each new sprite appears when it is created.
4. BONUS: Assign each clone a different value for some properties, such as speed, color, or size. Try controlling these values from the master sprite rather than having each clone choose its own.

Grading Scheme/Rubric

Lab 5.2 Criteria	
1.1 One bouncing sprite	0.5 points
1.2 Sprite controlled by stage	0.5 points
2.1 Clones created by stage	0.5 points
2.3 Hide master sprite	0.5 points
2.4 Bonus: Sprites have different properties	0.5 points
PROJECT TOTAL	**2.5 points**

Lesson 5.3: Communicating with Clones

Learning Objectives

Students will be able to...

- Pass information from the master to individual clones
- (Optional) Describe a race condition that might occur due using global variables and clones
- Delete clones when they are no longer needed

Materials/Preparation

- ☐ Do Now 5.3: Star Wars Troopers using Cloning
- ☐ Lab 5.3 Handout (Fewer Balls) (Download in Word) (Link to PDF)
- ☐ Test out the lab on student machines before class to ensure the machines can handle the number of clones
- ☐ Unit 5 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introductory discussion
30 minutes	Fewer Balls lab

Duration	Description
15 minutes	Debrief and wrap-up

Instructor's Notes

1. Introductory discussion

- Review the lab from yesterday again, focusing on the identical nature of the clones
- Ask students to describe how (or if) the prototype communicates to the clones
- Ask students how they might remove balls if they decide there too many
- Students should ultimately realize that there is no way to destroy only some clones
- Introduce the distinction between "global variables" and "sprite variables"
- Global variables ("for all sprites") are visible to and usable by all sprites in the program
- Sprite variables ("for this sprite only") are only visible to and usable by a single sprite
- Emphasize that, when cloning is used, each clone gets its own copy of any sprite variables inherited from the prototype
- Ask students to brainstorm situations in which each type of variable is appropriate
- Global variables are best for application-level data, such as sprite counts, game level number, score, etc.
- Sprite variables are best for properties that may be specific to each sprite, such as speed, id #, etc.

2. Activity

- Students should complete the Fewer Balls lab.
- Students will likely have difficulty isolating the uses for each variable. Remind them that each variable serves a specific purpose, and that they should focus on keeping straight what variable does what.
- Currently, the lab is written to use global variable "id" to talk to one sprite at a time. If not used carefully, this design pattern has the potential for race conditions. If you believe students are capable, you can have a discussion about race conditions and concurrency and the problems that can arise.

3. Debrief

- Have students pair up and review each other's code. Encourage students to discuss differences in their approaches and try to understand why each wrote the code as they did.
- If students have struggled with the lab, consider implementing helping trios
- Ask a few students to share difficulties they or their partner had and/or different approaches they took the problems.

Accommodations/Differentiation

- For faster students, explaining and demonstrating race conditions would be valuable. Instructors could also give out project with a "simple race condition" (if such a thing exists) and ask the students to fix it
- Advanced students can also try to implement features giving them additional control over clones, such as changing a specific clone's speed or direction.
- Struggling students can ignore the requirement to be able to create new sprites after some have been deleted and justfocus on deleting clones one at a time.

Forum discussion

Lesson 5.3: Communicating with Clones (TEALS Discourse account required).

TEALS Unit 5 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 5, 9, 12, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Cloning	A feature that allows a sprite to create a clone, or semi-duplicate, of itself, while the project is running. Clones of a sprite will be the same as the original or parent sprite but as a separate instance. Clones inherit the parent's scripts, costumes, sounds, and properties, yet they can then be modified.
Global Variable	A variable that can be used by all of your sprites.

Do Now 5.3 Star Wars Troopers using Cloning

1. Yesterday we started the day using duplicate to program 5 Star Wars troopers to march across the stage. Today we will use Snap's clone feature.
2. Go to this SNAP project. Using the following "create a clone of" block, create a script so there are 5 clone troopers that move from the right side of the screen to the left side in a straight line when green flag is clicked. The starting and ending positions of the sprites are shown below.

   create a clone of

Start

clone formation

End

clone formation end

Lab 5.3 - Fewer Balls

In this lab, you will build on what you created in lab 5.2 to enable better management of the number of sprites in the program.

Part 1 - Getting Out of Hand

1. Open up your SNAP program from Lab 5.2. Modify your program so that, when the 'd' key is pressed, all bouncing sprites are deleted. DO NOT delete the prototype– you should be able to create new bouncing sprites . DO NOT delete the prototype– you should be able to create new bouncing sprites after you have removed the old clones. Use the delete this clone block block and a message.
2. What if you wanted to remove only a few clones? Or only specific clones? What would be needed in order to accomplish that?

Part 2 - Better Control

1. Add a **global** variable to your program called `g_nextID` and give it a value of 1 when the green flag is clicked.
2. In your master bouncing sprite, create a **sprite** variable called `s_ID`. Modify your program so that each time a new clone is created, the clone's `s_ID` variable gets the value currently in `g_nextID` and `g_nextID` is incremented by 1.
3. Change your program so that when the 'd' key is pressed, the *newest* bouncing sprite gets deleted. Think about the right way to use the variables you created in the previous steps to know which sprite to delete. (*Hint: in order for this to work right, you should reuse old IDs once the clones are deleted. So, for example, if the most recently created clone was number 6, and you hit 'd', clone number 6 should be deleted. Then, if*

a new clone is created, it should be a new clone number 6.) Try to do this without requiring a lot of special cases in your code– every clone should operate in the same way to determine if it should be deleted.

4. BONUS: Add code so that if the 'x' key is pressed the program asks for an ID number and deletes that numbered clone. All clones with higher numbers should be renumbered so that ID numbers remain contiguous. (For instance, if clone number 5 is deleted, then clone numbers 6, 7, and 8 should be renumbered as clone numbers 5, 6, and 7 respectively. Then, the next clone created should be a new clone number 8.) This is tricky and will require you to think very carefully about how to use the variables.

Grading Scheme/Rubric

Lab 5.3 Criteria	
1.2 Delete all clones	0.5 points
2.1 Assign unique s_ID to each clone	0.5 points
2.2 Delete newest clone	0.5 points
2.3 Delete newest and add combined	0.5 points
2.4 Bonus: Delete specific clone	0.5 points
PROJECT TOTAL	**2.5 points**

Unit 5 Quiz Cloning

Learning Objectives

Formative assessment on student progress: To gauge student understanding, the addition of Unit quizzes has been added. These are intended as low stakes formative assessments that allow students to visit topics at the end of the unit to reinforce learning. They are open book giving students incentive to take good notes. Ideally the quizzes are non-graded and students would reflect on the answers they got wrong in order to learn from their mistakes.

Materials/Preparation

- TEALS Classes can access the Quizzes by logging into the TEALS Dashboard and navigating to "Additional Curriculum Materials" -> "Intro CS Curriculum".
- You will need to log in using incognito mode on the browser.
- See Additional Curriculum Resources for further instructions.

Lesson 5.4: Space Invaders Project

Learning Objectives

Students will be able to...

- Use cloning to implement a complete version of "Space Invaders"
- Exercise good programming practices to produce code that is not only functional but also elegant and well-written

Materials/Preparation

- Project 5 - Space Invaders (Download in Word) (Link to PDF)
- [Optional] Printouts of the specification
- Example ofSpace Invaders
- Unit 5 Tips

Pacing Guide

Duration	Description
Day 1	
5 minutes	Welcome, attendance, bell work, announcements
30 minutes	Review unit concepts
20 minutes	Introduce project
Days 2-12	
5 minutes	Welcome, attendance, bell work, announcements
10-15 minutes	Review
30-35 minutes	Lab time
5 minutes	Exit ticket

Instructor's Notes

1. Review/Introduction

- Play a review game (such as GrudgeBall) to remind students of the skills and concepts have been learned in this unit.
- Definition of "prototyping"
- Creating clones
- Using a master sprite
- Modifying clone behavior
- Passing information from the master to clones
- Deleting clones
- Remind students that their solutions to previous assignments are an excellent resource when trying to accomplish similar tasks.

2. Introduce project

- Walk students through the project specification, pointing out important details, potential pitfalls, and requirements.
- If students are unfamiliar with Space Invaders, spend a couple minutes demonstrating one for the class. If you have a SNAP or Scratch version, that works best, but an online game will work as well.
- Remind students that their version of the game does not need to exactly mimic the classic version. In particular, they need not have quite as many invaders, create barriers for the player to hide behind, or duplicate invader movement or firing patterns exactly.

3. Project

- This project is a summative assessment for the unit. Students should be demonstrating mastery of all the skills covered.
- Most students will require roughly 10-15 hours of total work time to complete the project
- Assess the progress of your students regularly using such techniques as asking them to demonstrate their incomplete programs, tracking questions asked during lab time, and/or utilizing peer reviews.
- Adjust the amount of time allowed for the project to fit the needs of your students
- It is vital that nearly all students complete the project before moving on
- If most students have the ability to work on SNAP assignments at home, the amount of in-class time provided can be reduced if necessary.
- If this approach is taken, be sure to make accommodations for students who are *not* able to work at home, such as after school lab hours
- Ensure that students are able to ask questions in class throughout the project
- See the standard Lab Day Lesson for detailed plans for lab days.

Sample Solution

- Sample project solution

Accommodations/Differentiation

- If any students do not have the ability to work at home, ensure enough in-class time is provided to complete the assignment, offering extensions if necessary.
- Advanced students can be encouraged to add different types of invaders that behave differently, implement a Galaga-style "swoop" behavior from the invaders, add player power-ups (such as advanced weapons), or any other extension.
- Struggling students can be exempted from certain features (such as levels or invader firing) or given starter code
- If students need significant assistance, focus them on the invaders. Getting a set of invaders created and moving properly will best target the vital objectives from this unit– namely cloning.

Forum discussion

Lesson 5.4: Space Invaders Project (TEALS Discourse account required).

TEALS Unit 5 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 3, 5, 9, 12, 17

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Cloning	A feature that allows a sprite to create a clone, or semi-duplicate, of itself, while the project is running. Clones of a sprite will be the same as the original or parent sprite but as a separate instance. Clones inherit the parent's scripts, costumes, sounds, and properties, yet they can then be modified.
Global Variable	A variable that can be used by all of your sprites.

Project 5 - Space Invaders

Students will implement a SNAP version of the classic arcade game *Space Invaders*.

Overview

Originally released in 1978, Space Invaders was one of the modern video games and is often credited with popularizing the video game industry. In addition to being a vastly popular arcade game, the 1980 version for an early Atari system helped game consoles achieve the mainstream status they maintain today. Because of this, the characters and gameplay style of Space Invaders are often used to represent the video game industry as a whole. You can play an online version. Our game will have some differences from the classic version, but will maintain the key aspects of gameplay.

Behavior

Sprite

Your implementation of Space Invaders will include at least three main sprites: the **player**, **invaders**, and **projectiles**.

- The **player** moves along the bottom of the stage attempting to destroy invaders by firing projectiles. The player can move left and right only (controlled by the arrow keys), not up and down. The player fires when the space bar is pressed. The player begins with three lives and loses a life each time an invader touches him or reaches the bottom of the stage.

- **Invaders** begin at the top of the stage and slowly move towards the bottom in a grid formation. Invaders move from side to side across the stage, bouncing off the left and right edges. Each time an invader bounces off an edge, it should move down by the height of one invader. When an invader is hit by a projectile, it is destroyed, and should disappear from the stage.
- **Projectiles** are fired by both the player and the invaders. The player fires a projectile when the user presses the space bar, while the invaders fire randomly. A projectile should originate at the position of the sprite that fired it and move quickly up or down the stage until hitting either an invader, the player, or the edge of the stage. If the projectile hits an invader or player, that invader or the player should be destroyed. Upon hitting either an edge of the stage or another sprite, the projectile should then disappear. Only one invader may fire a projectile at time, and another invader may not fire until the projectile has disappeared. The player has an unlimited supply of projectiles, but may only shoot *three* at a time.

Starting the Game

When the green flag is clicked, the game should initialize and a welcome screen should be displayed to the user (think of this as inserting a coin into an arcade machine). The welcome screen should include a "Start Game" button that, when clicked, begins the actual game. The welcome screen should also display some basic information about the game.

Lives and Game Over

The player begins the game with three lives, and loses a life each time an invader either touches the player or reaches the bottom of the stage. When the player loses a life, the game should halt momentarily. All invaders and projectiles should disappear, and there should be a visual indication that the player has lost a life (message, change in costume, etc.). After a brief delay, the game should restart with all invaders at the top of the stage and the player having one fewer life. When the player loses its final life, the game ends and a "Game Over" message should be displayed. A "New Game" button should then be displayed that, when click, restarts the game from the beginning.

Levels

Each time the player manages to destroy all the invaders on the screen, he or she completes a level. When a level is completed, a "Level Cleared" message should be displayed. Then, after a brief delay, the next level should begin. Each successive level should be made more difficult by increasing the speed at which the invaders advance down the stage and the frequency at which they fire projectiles.

Score

Each time the player destroys an invader, points should be earned. Scoring should start at 100 points per invader in the first level, and increase by 50 points per invader in each successive level. So, for example, each invader in level 2 is worth 150 points, each in level 3 is worth 200 points, at so on. The player's score should be displayed on the stage throughout the game and be featured on the "Game Over" screen.

Implementation Details

Documentation and Style As with all previous projects, your program must be well-written, well-documented, and readable. This includes, but is not limited to:

- organizing your scripts so that they can be read and comprehended easily
- giving your sprites meaningful names
- naming and using your variables, lists, and custom blocks well
- including comments to describe the structure of your program and any particularly complex or unintuitive pieces of code
- separating master sprites from clones, and cleaning up clones that are finished (see below)

Cloning The invaders and the projectiles should be implemented using prototyping. A single master sprite should be implemented for each type and clones should be created each time a new instance of the sprite is needed. The master sprites should be hidden throughout the program and should not take part in gameplay.

Required SNAP Elements Your program must include the following SNAP elements: * At least three variables * At least two custom blocks, at least one of which must take an argument

Required Checkpoints

1. Have the player and a single invader moving correctly
2. Be able to clone invaders and have the entire group move correctly
3. Be able to fire projectiles from both the player (when the space bar is pressed) and the invaders (randomly)

Grading Scheme/Rubric

Functional Correctness (Behavior)	
Welcome screen containing game info displayed green flag clicked	1 point
Start button displayed and game starts when clicked	1 point
Keyboard control of player (including shooting)	1 point
Multiple invaders	2 points
Invaders attack in formation	2 points
Invaders die when hit by projectile	1 point
Player earns points when an invader is killed	1 point
Player fires projectiles correctly	2 points
Invaders fire projectiles correctly	3 points
Player loses life and level resets when player or bottom of stage touched by invader	2 points
Game ends and new game screen appears when player is out of lives	1 point
Level changes when all invaders are destroyed	1 point
Invaders speed up in each level	1 point
Point values increase each level	1 point
SubTotal	*20 points*
Technical Correctness (Implementation)	
Program is well-documented and shows good style	2 points
Program shows creativity and effort	2 points
Invaders and projectiles are implemented using cloning	3 points
Program operates correctly under all normal conditions (aka "No bugs")	3 points
Program uses at least three variables	2 points
Program includes two custom blocks, one of which accepts arguments	3 points
Checkpoint 1	5 points
Checkpoint 2	5 points
Checkpoint 3	5 points
SubTotal	**30 points**
Total	**50 points**

Unit 6

Lesson 6.1 - Design Basics

Learning Objectives

Students will be able to...

- Identify the key considerations when designing a piece of software

- Describe methods for prioritizing features, use cases, and/or scenarios
- Explain why design and planning are necessary steps in the software engineering process

Emphasize with students

Curricular Competencies - understanding context, defining, ideating In this course, we are not actually typing out any code. In our case, "coding" would refer to the stage where we actually drag-and-drop in the SNAP design environment. Later when we learn other languages, "coding" would refer to when we start typing out code in a specific language like Java or Python.

When we first learn computer programming, it may seem fun to dive right into coding. Resist the urge to do so! Software development is a process. It begins with carefully thinking about, and planning out, your design. This includes envisioning what the final product might look like, and planning out the steps needed to implement it. Remember, "design then code".

Materials/Preparation

- Unit 6 Tips
- Final Project Plan Organizer
- Final Project Development Plan
- Examples of TEALS Final Projects

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Introduce final project, demo examples
25 minutes	Sample Design activity
10 minutes	Debrief and wrap-up

Instructor's Notes

1. Introduce project

1. Talk about how far students have come this semester
    - Ask students to think back to the start of the semester and remember how little they knew about programming.
    - Briefly list a bunch of the things they've learned (drawing, animation, variables, loops, conditionals, loops, etc.)
    - Maybe show a lab or assignment from early on and remind them that, not that long ago, this was challenging, whereas it now seems nearly trivial (hopefully).
2. Explain that, for their final project, the students will get to design and build a program of their own choosing.
    - Point out that this will involve more than just writing code– there will be planning, design, scheduling, and other project management tasks
    - Emphasize that students will be graded on not only the program they produce, but the process they used to design, plan, and implement it
3. Demonstrate a few example projects (with as much variety as possible).
    - Try to hit a bunch of different types of programs. Many students will gravitate towards games, but other options include simulations, productivity tools, musical projects, animations, and more.
4. Distribute the project rubric and point out key aspects of the requirements
    - Point out the steps in the process and that each one is equally important
    - Specifically mention the large number of points for things *not* related to coding

- Remind students that, as they are now SNAP experts, there are high expectations for the depth, complexity, and completeness of their projects

There are a gazillion apps out there to do a gazillion things. What are the different categories of software applications? When looking at an application, what type of functionality does it have? Who are the end users, or target audience? What do you like, or not like, about it? What do other users/reviewers say about it? How does it compare to, or stand apart from, other similar apps? As we study and analyze sample applications, we gain experience that will help us design our own.

2. Sample design activity

1. Tell students that, as a class, they will now practice some of the design and planning tasks for the project on a well-known app
    - Pick an app that both you and most of your students have a deep familiarity with, but that is not too complex. Twitter, Instagram, Pinterest, or other relatively small-scope social media apps work well here.
    - If necessary, scope down the app by focusing on only the core features (e.g. only consider public, text-only Tweets to start)
2. Walk through each of the design steps in the project rubric for the chosen app:
    - **Pitch** - describe the basic functionality of the app in one paragraph of less
    - **Define** - List the features/scenarios the app will support
    - **Sketch** - Draw a very basic wireframe sketch of the main "screens" of the app
    - In this step, try to keep students from fixating on making their app look *exactly* like the existing app
3. Expand - Build a spec using the Final Project Plan Organizer.
    - In this step, emphasize completeness and detail. Leaving out steps or requirements will make it difficult to plan effectively and will likely force major changes or cuts later.
4. Plan - Based on the feature list and spec, create a full development plan using the Final Project Development Plan.
    - As tasks and costs are listed, remind students to keep an eye on the total amount of time required and to include buffer for things going wrong. Be sure to prioritize tasks so that cuts can be made if necessary.

3. Debrief

- Once all documents have been created, ask students what the next steps should be.
- Get more detail than "start coding." Students should be focused on the highest-priority tasks and should understand the plan for evaluating progress throughout the project.
- Preserve the documents created in the activity to give to students as examples when planning their own projects.
- Emphasize that both the spec and the plan are "living documents" and should be updated as the project progresses (when new features are thought of, tasks are cut for time or complexity, priorities are reorganized, etc.)

Accommodation/Differentiation

- This lesson could easily take two or even three days depending on the class. Take as long as is necessary to ensure students have a complete understanding of the steps necessary to create their final project, but try not to get too bogged down in the details of the sample app.
- For classes with a large number of self-sufficient students, consider having students complete this activity in small groups rather than as a full class. Along the way, have groups share their progress and provide feedback to each other.

Forum discussion

Lesson 6.1 - Design Basics (TEALS Discourse account required).

TEALS Unit 6 Tips

SNAP Tips

Tip Numbers from SNAP Tips Document: 0, 21, 23

Word Wall

Terms introduced in the unit that you may consider adding to a classroom Word Wall.

Word	Definition
Scenario	A description of a set of interactions and/or tasks that describe a start-to-finish example of how a user might want to use the application
Wireframe	A high-level sketch of an application's user interface intended to help visualize layout, interactions, and transitions

Lesson 6.2 - Research and Ideate

Learning Objectives

Students will be able to…

- Identify potential users, intended impact, and possible unintended negative consequences
- Generate ideas to create range of possibilities using a brainstorm technique
- Conduct user-centered research to understand design opportunities and barriers
- Critically analyze factors when choosing between project ideas
- Prioritize proposed project ideas using the identified factors

Emphasize with students

Big Ideas - personal design interests and project goals When coming up with ideas, personal interests are considered. The final product often reflect the "heart and soul" of the designers behind the product. If you are working in a group, each person should be willing to see the project through to completion.

Big Ideas - product life cycle The design of a new software product progresses through a sequence of stages called the "life cycle", and is often associated with changes in the marketing situation. The first step of introducing a product involves getting to know the market, or in other words, the users or potential customers out there.

Curricular Competencies - understanding context, defining, ideating When coming up with a great product idea, it is important to conduct user-centered research to understand the needs of the client/customer/end-user. This research helps us understand design opportunities and barriers. This research could be done as a survey or interview, given to random people, to people in your friends or family circle, or to a specific population of people (like customers of a supermarket, or members of a soccer club, for example).

Materials/Preparation

- [ ] Unit 6 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review process and identify first steps
5 minutes	Brainstorming
10 minutes	Pitch writing

Duration	Description
15 minutes	Peer review
5 minutes	Prepare interview or survey questions
5 minutes	Debrief and wrap-up

Instructor's Notes

1. Review

- Ask students to identify the steps in the design and planning process as discussed in Lesson 6.1.
- Remind students that all steps are vital, and that thorough and thoughtful planning and design can make the coding phase much easier.
- Inform students that today they will take the first steps in designing their final project.

2. Brainstorming

- Give students 3-4 minutes to brainstorm and write down as many project ideas as they can. This should be done mostly in silence.
- At this point, there should be minimal detail, no evaluation or rejection of ideas, and no discussion. In particular, students should not think about the difficulty or "coolness" of the project yet. Just write down ideas.
- If desired, have each student share one idea. Do not allow discussion, criticism, or explanation– each idea should be summarized in only a few words or a single sentence.

3. Pitch writing

- Have students look at their list of ideas and spend a few minutes thinking about them. Then, each student should pick their favorite ideas and write a "pitch" for the project. A pitch should be no more than a short paragraph and should describe the basic, high-level features of the project. The pitch should *not* include any implementation details (scripts, sprites, etc.).
- Pitches should include a moderate level of specificity– enough for someone to imagine how the app will work, but not so much to get bogged down. Enforce the "one short paragraph" restriction.
- If a student is having difficulty developing a pitch for an idea, that might be a sign that the idea is not fully-formed enough to be a final project.
- If a student is having trouble keeping the pitch short, the project may be too complex to complete in the available time.

4. Peer review

- Pair students up and have students take turns reading one of their pitches to their partner and asking for feedback. Partners should ask questions to help identify both the best and worst parts of each pitch.
- Remind students to keep all feedback constructive, respectful, and professional. Students should not criticize each other's ideas, but can point out potential concerns.
- Students should take notes during their conversations and refine their pitches based on their partner's feedback and their own realizations.
- If time allows (or over the course of multiple days), repeat this process with new partners.

5. Prepare interview questions, and evaluate

- At this points, students should have between one and three pitches that are well-defined and reasonably well fleshed-out. Overnight, students should do some research, re-consider their pitches and rank them in order or which they would most like to pursue as their final project.
- Make sure students don't just pick the "coolest" sounding idea, but also consider the technical challenges, amount of time available, and their own interest in and willingness to see the project through to completion.
- As homework, students may create a simple survey or a list of questions to ask friends or relatives to help refine ideas, and subsequently, decide on the final project. Make changes to the project pitch based on these recommendations.

Software design is a process that takes time. Conducting user-centered research helps narrow down and refine ideas. What do potential users think? For example, what components are necessary? What components are nice to have? Is it easy to use?

Accommodation/Differentiation

- If students are having difficulty coming up with project ideas, encourage them to think about existing software. While simply recreating an existing app should be a last resort, thinking about apps they already know can help students come up with functionality they might like to include.
- If your class is fairly self-sufficient and mature, you can consider allowing students to "borrow" an idea from a classmate if they find one they like better than any of their own. Make sure the person who had the idea is OK with it being borrowed, and emphasize that the students must each build their own version.
- This can be a bit dangerous, as it puts the student somewhat behind in the process right from the start, since they won't have as much context having not written the pitch themselves. Consider carefully whether this is a good idea.
- Encourage each student to pick a project that fits his or her level of technical skill. Make sure students are not overcommitting and choosing projects they do not have the skills to complete. Also try to discourage stronger students from choosing simpler projects in an effort to do less work.

Forum discussion

Lesson 6.2 - Brainstorming and Evaluating (TEALS Discourse account required).

Lesson 6.3 - Defining Requirements

Learning Objectives

Students will be able to...

- Define key **user-scenarios** for a project and the features required to implement each scenario
- Explain the importance of wireframing when designing an application
- Capture key scenarios using **sketches** - (hand drawn, or with drawing tool)
- Refine design based on user-centered research

Emphasize with students

Curriculum Competencies - Prototype, Test, and Make The term "Agile Development" is used in software development to describe an iterative and incremental project management process, involving short cycles called "sprints". Each sprint involves defining a list of tasks, doing a set of tasks, testing, going back to review the tasks list, and refining the design as needed.

From Lessons 6.3 to 6.5 you will be doing this type of iterative and incremental work. Lesson 6.3 does not involve any "coding" yet. You will start with sketching things down with pencil-and-paper, or with a drawing tool. Use the handouts to help you. You may revise some sketches later on, after you have started implementation (ie, "coding") and testing.

Materials/Preparation

- ☐ Final Project Plan Organizer handout
- ☐ Optionally, students can keep a daily journal notebook

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review pitches
20 minutes	Scenario definition

Duration	Description
15 minutes	Wireframing
5 minutes	Debrief and wrap-up

Instructor's Notes

1. Review pitches

- Have students look at their revised pitches from yesterday. Tell them that they will be choosing **one*- to pursue as their final project.
- If desired, give students a few minutes to rework their pitches or get more feedback from a classmate or instructor.
- Ask students to choose which idea they want to pursue, and write it down on the top of their Final Project Plan Organizer.

2. Defining Scenarios

- Defining the term "scenario" for students:
- *scenario: a description of a set of interactions and/or tasks that describe a start-to-finish example of how a user might want to use the application*
- Explain that defining scenarios helps a programmer focus on what features are actually necessary to enable the key user interactions for their application
- Instruct students to write down at least **3*- scenarios for their project describing, from start to finish, interactions a user might have with their program to accomplish a specific goal
- The scenarios should have a moderate level of detail in the description of the user interaction (e.g. "push a button," "type in their name," etc.) but should not include any design or implementation details.
- Once students have written their scenarios, they should review them and develop a list of the necessary features to enable each scenario
- Again, there should be minimal technical detail in these descriptions, instead focusing on details of the user experience. The feature lists should be more about *requirements* than implementation.

3. Wireframing

- Define the term "wireframe" for students:
- *wireframe: a high-level sketch of an application's user interface intended to help visualize layout, interactions, and transitions*
- Explain that wireframes do not include any details (such as specific graphics or text), but instead provide a broad impression of what an application will look like to aid in design and planning
- Students will complete page 1 of the organizer by sketching or wireframing the important screens for their project.
- If you have one available, a wireframing tool such as Balsamiq can be used instead of sketching by hand.
- As with all wireframing, students should not focus on the specific details of the sprites, images, etc. that will appear, but instead design the basic layout and main components of each screen.
- Encourage students to reference their feature lists to ensure they include *all* necessary screens for their project, including simple things like a splash screen, help screen, or exit ("game over") screen.

4. Debrief

- As class ends, ensure students retain their work as they will use it to construct a detailed specification and implementation plan tomorrow.

Accommodation/Differentiation

Forum discussion

Lesson 6.3 - Defining Requirements (TEALS Discourse account required).

Lesson 6.4 - Building a Plan

Learning Objectives

Students will be able to...

- Break down, and identify the main technical components needed for the functional project specifications (scenarios)
- Explain the purpose of each technical component
  - Develop the project idea into a full, detailed specification
  - Create a plan that includes specific steps, ie, list of bite-sized tasks
  - Estimate time required for tasks

Emphasize with students

Curriculum Competencies - Prototype, Test, and Make From Lessons 6.3 to 6.5 you will be practicing an "Agile" iterative and incremental project management workflow. In Lesson 6.3 you identified user scenarios. Now in Lesson 6.4 you identify what technical components are needed for each scenario, and define all the tasks that need to be done. You are not "coding" yet.

Each defined task, or step, should be bite-sized, and specific enough to do.
This plan should include testing done by you, as the developer, and by other users. As you progress, gather feedback from users/testers over time, and make changes to your design, and plan, as needed.

Curriculum Competencies - Applied Technologies Your classroom may have a diverse student population, and include English Language Learning (ELL) students.
Following an Agile project management flow helps ensure that the development process is simple, clear to follow, well documented, and easy to share and evaluate. Practicing these skills will help students collaborate and share their work in diverse classrooms. These skills prepare students to engage more successfully in an increasingly global technology industry.

Reference

Agile Project Management English Language Learning

Materials/Preparation

- ☐ Final Project Plan Organizer handout
- ☐ Final Project Development Plan handout
- ☐ Unit 6 Tips

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Review feature lists and wireframes
20 minutes	Spec writing
15 minutes	Building implementation plan
5 minutes	Debrief and wrap-up

Instructor's Notes

1. Review

- Ask students to take out their feature lists and wireframes from Lesson 6.3. They will be using these to develop a more detailed specification and plan for their project today.
- If time allows, ask one or two students to share their feature list and/or wireframe and discuss with the class.

- Ensure that students have an understanding of the proper level of detail at this point.

2. Spec writing

- Using the details from their pitch, their feature lists, their wireframes, and the feedback they've received, students should fill out the rest of the [plan organizer].
- It is **VITAL*- at this stage that students be as detailed and thorough as they can. Any missing information will complicate the process later when they realize what was left out. Encourage students to take their time and make sure they hit everything.
- While this process is happening, instructors should circulate through the class and check-in with student. Verify that they have a complete, well-thought out idea that is feasible to complete in the available time.

- If you have concerns about a student's ability to complete the proposed project, help them scope down by removing or simplifying features.

3. Implementation plan

- Students should use the details built in their plan organizer to list the tasks necessary on their Final Project Development Plan.
- Emphasize to students that tasks should be at a very low level of granularity (hence the time requirement being specified in minutes). If a single task has a time estimate of more than a few hours, the student should try to break the task into smaller pieces.
- Ensure that students do not skip "trivial" or "simple" tasks (such as building a script they have written before) or non-coding tasks (such as developing graphics) in their plan.

4. Debrief

- As class ends, remind students that their spec and implementation plan will be their guides throughout the process. They should update them each day and keep them with them at all times.
- Ideally, anytime there is a question about the requirements or scope of the project, the spec should have the answer. If not, it's a new idea and the spec needs to be updated accordingly.

Accommodation/Differentiation

Forum discussion

Lesson 6.4 - Building a Plan (TEALS Discourse account required).

Lesson 6.5 - Project Implementation

Learning Objectives

Students will be able to...

- Use the skills developed throughout the course to implement a medium- to large-scale software project
- Realistically evaluate progress during software development and identify when cuts are necessary
- Prioritize features and scenarios and choose which should be eliminated or modified if/when resources and/or time become limited
- Record time taken for tasks, and lessons learnt in the process, to help refine estimates
- Record iterations of prototyping

Emphasize with students

Curriculum Competencies - Prototype, Test, and Make From Lessons 6.3 to 6.5 you will be practicing an "Agile" iterative and incremental project management workflow. Now it's time to start following your plan. You can finally start "coding", and testing! Remember that not all tasks involve coding. Take note of the time it takes you to complete each task (this will help you make better estimates in the future). Estimating time for tasks can be difficult to do. This will get better with experience.

Materials/Preparation

- ☐ Students should each have their Final Project Plan Organizer and Final Project Development Plan

Pacing Guide

Duration	Description
Days 1-15	
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Check-in
30 minutes	Lab time
10 minutes	Exit ticket

Instructor's Notes

1. Check-in

- Remind students daily to keep their planning documents up-to-date and make edits as necessary.

- Point out how many days remain and have students check their implementation plan to ensure they do not have more work than time remaining.
- If they do, they will need to create a tentative cut list in case they don't catch up.
- Using previous days exit tickets, questions from students, instructor awareness of trouble points in the project, and/or any other resources to determine what needs covering
- Use this time as an opportunity to remind students of previous labs or activities that may be applicable to their project, and/or how far along they should be by the end of the day

2. Lab time

- Allow students to work on their project at their own pace
- Provide a mechanism for students to ask questions of course staff as needed
- Simply having students raise hands often does not work well, as it can be hard to keep track of in what order hands were raised; consider a queue of some kind where students write their names when they have a question
- When there are no current questions, circulate and observe progress, stepping in if students appear stuck but are not asking for help
- Be sure to meerkat and not spend more than a minute or two with any single student at a time

3. Exit ticket

- Before students leave, have them answer the following questions on a small piece of paper, or in their daily journal notebook:
  1. What was the last thing you accomplished on the project today?
  2. What is the first thing you will work on tomorrow?
  3. Are you currently ahead, behind, or on track with your schedule? If you are behind, what tasks will you cut to get back on track? If you are ahead, what are some extra features you can add?
  4. What is the riskiest remaining task for your project?

- These answers will help you determine which students to visit first the next day.

- Any student who indicates they are behind should get a consult with an instructor the next day to help get them back on track.

- Encourage students to save each day's version of their planning documents with a new name (possibly using the suffix "_mmdd") so they can track progress and recover cut tasks if they make up time.

Accommodation/Differentiation

Forum discussion

Lesson 6.5 - Project Implementation (TEALS Discourse account required).

Project 6 - Final Project

Students will design, plan, and implement a medium- to large-scale final project of their own choosing.

Overview

During this course, you have learned a huge amount about computer science and programming in general, and SNAP in particular. In this project, you will put all of that knowledge, along with some new skills you will develop around design, planning, and project management, to build a relatively large and complex application that *you* choose. You can create almost anything you want and should ultimately produce a project that is interesting, useful, and challenging.

Details

Project Phases

This project will be significantly larger in scope than any of your previous assignments, so there will be more design and planning than before. More importantly, though, rather than be given a well-defined specification, *YOU* will be setting the requirements for your project by coming up with an idea, fleshing out the details, and defining the steps necessary to complete your program.

To help you through this process, there will be several steps to this project. You must complete **all** of the steps **in order** for your project to be successful. In fact, *half* of your grade will be based not on how well your program works, but on how well you completed the design and planning process.

The phases of the project will be:

- *Brainstorming* - coming up with as many possible project ideas as you can
- *Pitching* - choosing a few ideas and developing a short description of what the project will entail
- *Review* - getting feedback from your peers and instructors on your pitches and choosing one
- *Scenario Definition* - listing out the features the project will need and what they will look like
- *Wireframing* - drawing simple sketches of what the various "screens" in your program will look like
- *Specification* - fleshing out all the specifics of how the project will work
- *Scheduling* - listing the programming tasks necessary to complete your project and estimating how long each will take
- *Development* - writing the code for your project by following the spec and schedule created in the previous steps

Progress Tracking

In phase vi, you will complete a Final Project Plan Organizer and in phase vii you will complete a Final Project Development Plan. These documents will be your guides in the development phase and will help you stay on track and aware of your progress. Throughout the development phase of the project, you will be expected to keep your spec and plan up-to-date and make adjustments as you get ahead or behind, as requirements change, or as tasks or features get reprioritized. At the end of each coding day, your spec and plan documents should be updated to reflect the current state of your project, and you will check in with an instructor at least once a week to make sure things are on track.

Implementation Requirements

Complexity and Creativity Your final project should be sufficiently complex and large-scale to push your limits as a programmer, but not so sophisticated that you are not able to complete it in the time allotted. The complexity in your project should come from the *design* and the *algorithms* and not from the *code*. (That is, you cannot meet the complexity requirement simply by writing a lot of code. Your code must be challenging or interesting

in some meaningful way.) In addition, you should not add complexity by introducing peripheral elements, such as graphics or sound effects. (Your program can certainly have these, but they will not be considered in determining the projects complexity.)
In addition, one of the main goals of this project is to allow you to unleash your creativity and allow you to create something of interest to you. To achieve this, your project must show some level of creativity or personalization that makes it your own. Simply creating your own version of some existing application will not fully meet this requirement.
For both the complexity and creativity requirements, you should talk to the instructors early and often to ensure your project is in line with our expectations.

Documentation and Style As with all previous projects, your program must be well-written, well-documented, and readable. Writing code with good style is always a good idea, but in a project of this size and scope, following style guidelines will help you keep your thoughts organized and make it easier to keep track of your progress, pick up where you left off each day, and find and fix bugs. In particular, though this is certainly not a comprehensive list, pay attention to the following:

- organizing your scripts so that they can be read and comprehended easily
- giving your sprites, variables, lists, and custom blocks descriptive and meaningful names
- using the right type of variable (global, local, sprite) for each situation
- including comments to describe the structure of your program and track your progress
- avoiding redundancy with good use of loops, custom blocks, and/or lists
- practicing good procedural decomposition and abstraction

Required SNAP Elements In order to show that you have fully mastered all the skills from the course, you project must include at least the following:

- a clear way to start the program, and clear prompts or instructions for any user interaction
- at least one loop, variable, custom block, and list, and more as necessary or appropriate
- each these must be used correctly and meaningfully– creating a list that contains a single element just to meet this requirement will not earn points
- at least one user interaction
- this can be prompting for information using ask, responding to key presses or mouse movements, or any other action that keeps the user involved

Required Checkpoints At least three times during the project period, and at least once each week, you should check in with an instructor to ensure that your project is on track, that you are meeting the project requirements, and that you have the answers to any questions that might have arisen during your work. The course staff will work with you to set up a schedule for these checkpoints, but it is **your responsibility** to ensure that the meetings take place.

Grading Scheme/Rubric

Design Phases	
Brainstorming	2 points
Project Pitches	6 points
Feature List	4 points
Wireframes/Sketches	4 points
Project Organizer (Specification)	8 points
Implementation Plan	8 points
Spec and plan are updated throughout project	8 points
Total	*40 points*
Implementation	
Project is appropriately complex and creative	8 points
Program is well-documented and shows good style	4 points
Program uses SNAP elements effectively, including all required elements	8 points

Design Phases	
Final product meets all requirements and goals laid out in spec	8 points
Checkpoint 1	4 points
Checkpoint 2	4 points
Checkpoint 3	4 points
Total	*40 points*
Total	***80 points***

Lesson 6.6 - Project Sharing

Learning Objectives

Students will be able to…

- Share their progress, invite feedback, collaboration, and if applicable, prepare a marketing pitch
- Decide on how and with whom to promote and share their project
- Critically evaluate the design process, their ability to work effectively, including the ability to implement project management processes
- Identify new design issues, including how they or others might build on their concept
- Identify and evaluate their skills, and things to learn in the future
- Analyze the role and impact of their project idea, and similar technologies, in societal change
- Consider how cultural beliefs, values, and ethical positions affect the development and use of technologies

Emphasize with students

Curriculum Competencies - Share, Applied Skills, Applied Technologies At the end of the day, software must connect with people. Now it's time to share your project with others!
It's also fun, and helpful, to see what others have been doing.

Furthermore, it's time to reflect on the whole development process, and share what you have learnt from this experience.

Invite questions and feedback from peers and guests. Celebrate your achievement. Consider ideas for improvement and future work. Think about how you could do things differently using other tools/strategies. Think about how this project might be adapted for, or influenced by, another culture, social situation, or target user group.

Materials/Preparation

Pacing Guide

Duration	Description
Sharing Day	
5 minutes	Welcome, introduction
40 minutes	Project Sharing (Format defined by teacher)
10 minutes	Concluding thoughts

Instructor's Notes

1. Preparation

- Choose a date for Project Sharing ahead of time, and inform students of the format.
- Check that each student's planning documents, and or daily journal notes, are up to date
- Some possible formats (depends on the size of your class):
- 40 minutes for individual presentations, followed by 10 minutes of "open-floor" time where students can roam around the classroom and look at each other's work science-fair style, and answer questions;

- Prepare a Project Walk-through video ahead of time, showcasing their project;
- Share in pairs, and then shift to the next table/chair round-robin, or jigsaw style;

- Conduct an interview of a few projects at the beginning of class over a period of time.
- This may be a good opportunity to invite guests or visitors to your classroom.

Peer feedback is valuable. It helps us grow. This can be done by having a few minutes for Q&A after the presentation. Or, peers can be invited to complete a short survey. Remember to put any emotions or hard feelings aside.

2. Sharing Content

- Inform students that their sharing should include: pitch or poster (highlight key features); demo (how it works); reflections on their development process (what went well? what didn't go well); lessons learnt; ideas for future work.
- Other questions to consider:
  - What skills did you use to complete this project?
  - What other skills do you wish you had?
  - How does your project/product compare to other ones that you have seen?
  - How does a project/product like yours impact societal change?
  - Do you foresee any unintended, or negative consequences, of it's use?
  - How might someone of a different culture, value or ethical position view your project/product?

3. Concluding thoughts

- Congratulate students on completion of project.
- Emphasize that while having a fun final product to show others is great, this Unit is also about practicing a project management flow, where learning from self-reflection and experience (both good and bad - even mistakes) is invaluable.

Accommodation/Differentiation

Students who are shy and afraid of public speaking may be given the option of preparing a "walk-through" video to show-case their project experience.

Forum discussion

Lesson 6.6 - Project Sharing (TEALS Discourse account required).

Accessing Additional Curriculum Materials

To access the "Additional Curriculum Materials", log into your TEALS Dashboard at https://tealsk12.org/dashboard.

1. The Additional Curriculum Materials are stored in a Microsoft Office 365 SharePoint site under the TEALS domain.
2. You will find the link, username, and password for these materials under the "Resources" heading on the dashboard.
3. If you are already logged into an Office 365 account (perhaps because you use Office 365 at school or work) you will run into an authentication conflict when you try to access this SharePoint site in step 1. In this case, you need to access the "Additional Curriculum Materials" SharePoint site using an Incognito or InPrivate browser session.
4. To open an Incognito or InPrivate browser session, right-click on the "Additional Curriculum Materials" link and select "Open link in Incognito/InPrivate window".
5. The website will ask you to log into Office 365.
6. Be sure to use the username and password listed on the TEALS Dashboard, and not the account you usually use to log in at work or school.

- You need to have cookies enabled to use the TEALS Dashboard site in general.

Contributing

Repository Location

The curriculum's source code is hosted on GitHub at: https://github.com/TEALSK12/introduction-to-computer-science

You can open issues, fork the curriculum, or submit pull requests to suggest changes.

Markdown style

To keep the curriculum's underlying markdown consistent, we use this markdown style guide. Since markdown is not a strict specification, there are a few options we choose to take from the style guide

- space-sentence:1
- wrap:inner-sentence
- header:atx
- list-space:mixed
- code:fenced

Updating GitBook with changes

If you make changes to file names and links, make sure to update the GitBook references in summary.md accordingly.

Lint tool

We use mdast-lint to enforce the above style. All submissions will be run through mdast-lint and free of any errors and warnings.

Curriculum style

How to write *Snap!*

`*Snap!*`

Snap! Code

Blocks and scripts should always be presented as they would be in *Snap!* as an image.

Good > Use the move 10 steps block to move your sprite.

Bad > Use the move block to move your sprite.

If it is absolutely necessary that a block not have a picture, wrap the name in blockquotes (e.g. `move 10 steps` block).

Creating new script images Use the "script pic…" feature to create new images. If you need the result of a reporter block, use shift-right-click to get the "script pic with result…" option.

If you are adding a block, the file name should follow the text of the block with lower camel case.

Good > move 10 steps -> `move10Steps.png`

Bad > move 10 steps -> `move.png`

Reusing existing script images Before you create new blocks images, check to see of blocks are stored in the curriculum in `/blocks`.

All script images should be stored in `/scripts`.

Vocabulary words

Labs

Lesson plans

Creating a pull request

Each pull request should have it's own branch. Here are a few examples of a proper pull request workflow

- http://codeinthehole.com/writing/pull-requests-and-other-good-practices-for-teams-using-github/
- https://github.com/skyscreamer/yoga/wiki/GitHub-Best-Practices
- https://www.thinkful.com/learn/github-pull-request-tutorial/

[2.1.2] - 2018-11-28

Unit	Change
Culture_day_lesson_b.md	Added Curriculum Competencies and Possible Topics
Culture_day_lesson_c.md	Added “My Skills and Interests Journal” culture day
Culture_day_lesson_d.md	Added “Interview with People in Technology” culture day
lab_11.md	Fixed Alignment
lesson_33.md	Fixed Spelling
lesson_61.md	Added content curriculum Competencies
lesson_62.md	Added content Learning Objectives; Big Ideas; Curriculum Competencies - understanding context, defining, ideating and Curriculum Competencies - understanding context
lesson_63.md	Added content Curriculum Competencies; Materials / Preparation
lesson_64.md	Added content Learning Objectives; Curriculum Competencies - Prototype, Test and Make; Curriculum Competencies - Applied Technologies; Reference
lesson_65.md	Added content Curriculum Competencies; Materials / Preparation
lesson_66.md	Added Project Sharing; Curriculum Competencies - Share, Applied skills, Applied Technologies; Materials / Preparation; Instructor's Notes; Curriculum Competencies - Applied Skills; Accommodation / Differentiation; Forum discussion
Project_1.md	Changed Content from Canadian perspective to US perspective; Added Content Big Ideas; Reference; Behavior; Implementation Details; Sharing, Curriculum Competencies design sharing
Project_2_alternative.md	Added the Whole Content and changed content to US perspective
Project_3.md	Added and Changed Content on Big Ideas from BC specific to US specific; Added content Curriculum Competencies - Design Sharing
Summary.md	Added Culture day C and D, added Project 2 alternative

[2.1.1] - 2018-08-21

Unit	Change
lab_22.md lesson_22.md SUMMARY.md +Lab 2.2 Yellow Brick Road.docx/.pdf	Change theme from brick wall to yellow brick road

[2.1.0] - 2018-08-16

Unit	Change
0.2	Toothbrush activity option added to Do Now World Cafe protocol added for Lab
0.4	Student Experiences Survey added and updated lab files uploaded
1.1	Diverse Grouping guidance added for lab Helping Trios handout added as Accommodation/Differentiation
1.2	Options added to Do Now and Activity Cold Calling Alternative protocol added
1.3	Geometry Cheat Sheet handout added
2.1	Do Now option added Geometry Cheat Sheet handout added
2.2	Lab challenge activity edited to personalize Diverse Grouping reminder Cold Calling Alternative added in debrief
2.3	An intro to conditionals video by Flocabulary added
2.4	Variable Boxes Unplugged Activity added to Do Now as a intro to Variables
2.5	Boolean Expression Unplugged Activity added as intro to Boolean Expressions Reminder about diverse grouping Geometry Cheat Sheet handout added
3.1	Lab edited to include options for personalization
3.2	Do Now option added Reminder about diverse pairing
3.3	Lab challenge edited to include personalization of No. 4 updated lab files uploaded
4.2	Do Now option added Lists Structure Handout Grammar Cheat Sheet handout accommodation added
4.3	Lab language change & uploaded updated files Diverse Grouping reminder
4.6	Hangman project changed to Word Guessing game with edited handouts uploaded Snowman Snap! project replaces hangman example
5.1	Diverse grouping reminder
5.2	New Space Invaders links added includes Vimeo link Lab now focusing on bouncing sprites instead of balls & updated files uploaded
5.3	Lab now focusing on bouncing sprites instead of balls & updated files uploaded Helping Trios handout added
6.1	TEALS final project examples added TEALS design steps handout added
6.2	Peer Feedback Handout added Inspirational software design video from Code.org added Diverse grouping reminder
6.3	Links to Final Project Development Plan & Organizer added
6.4	Links to Final Project Development Plan & Organizer added
6.5	Wise Feedback protocol added Meerkating guidance added Links to Final Project Development Plan & Organizer
Overall	CRTchanges aim to provide for student choice, voice and agency in the curriculum. They include options and resources to address different learning styles and to personalize to the cultural specifics of your classroom.
Overall	"Unit Tips" documents have been added for each unit with helpful Snap! shortcuts and teaching tips relevant to the unit. It also includes definition of terms introduced in the unit that can be used on a classroom Word Wall.

BJC Lecture Suggestions

Dan Garcia of UC Berkeley presents the Beauty and Joy of Computing

About the Lectures

Use

The TEALS' Introduction to CS Course is based on the The Beauty and Joy of Computing by Dan Garcia at UC Berkeley. However, the TEALS' curriculum varies greatly in content and scale, as it is aimed at High School students. This page outlines a series of video lectures from Dan Garcia's version of this course–however the videos are not applicable in their entirety. The lectures are mapped out below by lecture, subject and time in their entirety.

Some of the lectures (or sections of the lectures) will be useful background for teachers learning the course materials. Some sections of the lectures are useful as instructional tools for classroom instruction. It is notated if the lecture is useful for background knowledge for teachers/volunteers and/or for classroom instruction. The lectures that are directly relevant to lessons or labs are directly referenced *below* the "Instructor's Notes". The lecture videos are licensed under a Creative Commons License by UC Berkeley. - BJC Lecture 1: Abstraction - Basic concepts of the course: 0:00-7:00 - Introduction of Piazza: 7:00-8:25 - Abstraction: 11:40-15:40 - Generalization: 15:50-20:00 - Summary: 20:05-25:10

- BJC Lecture 2: 3D Graphics
  - SOPA & PIPA: 0:00-1:00
  - 3D Computer Graphics Explanation: 1:00-5:24
  - 3D Graphics steps outlined: 5:25-5:50
  - Modelling (Useful for Lab 2.5): 5:50-11:40
  - Animation (Uncanny Valley Explanation): 11:40-16:55
  - Procedural Based Motion (Lab 1.1): 16:56-20:00
  - Genetic Algorithms: 20:05-25:25
  - Lighting and Shading: 25:25-27:10
  - Rendering: 27:10- 30:55
  - Global Illumination: 30:55-34:21
- BJC Lecture 3: Video Games

- Demystification Lecture-Novel Interaction techniques(emotive systems) 00:00-2:30
- History of Video Games Overview (Platform Game Prep) 2:30-7:55 **Good for Classroom Instruction**
- Casual Video Game World(light weight) 8:00-10:40 **Good for Classroom Instruction**
- Core Video Game (heavy weight) 10:45-13:05 **Good for Classroom Instruction**
- 3D Computer Graphics 13:10-15:20
- Motion Capture(Hero Movement for Platform Game Lab 2.6) 15:25-17:30 **Good for Classroom Instruction**
- Artificial Intelligence (Enemy Logic for Lab 2.6) 17:30-19:40
- Video Games w/purpose (social benefits) 19:40-24:36 **Good for Classroom Instruction**
- Negative Aspects of Video Games(RSI, addition, violence) 24:40-28:00 **Good for Classroom Instruction**
- Glenn Sugden-Game Developer(History of VG Development & Industry) 28:03-40:53 **Good for Classroom Instruction**
- BJC Lecture 4: Functions
- SIRI-EVI 0:00-1:00
- Functions & Generalizations (Function Basics) 1:00-3:45
- More Terminology (Boolean etc) 3:47-6:10
- Types of Input (Sentences, words, characters, digits) 6:12-8:00

- Functions (Explanations of Use-can be tied in to loops, and inputs) 8:00-9:55
- MIT Scratch –> BYOB SNAP (Development of SNAP, DEMO) 10:00-11:30
- Functions-1 (BYOB-Custom Blocks) & Generalization 11:30-14:50
- Functions-2 (Join Block) Domain and Range 14:52-17:50
- Types of Blocks 18:15-19:45
- Recursion Preview 19:50-27:40
- Functional Programming Summary (Big concepts narrow down to functions) 27:40- End
- BJC Lecture 5: Programming Paradigms
  - Dilemma of Being a Cyborg 0:00-2:30
  - Programming Paradigms 2:30-3:50
  - Snap! BYOB (Hybrid) 3:55-4:45
  - Functional Programming (Cascading Values) 4:50-5:35
  - Imperative/Sequential 5:41-8:35
  - Object Oriented Programming (OOP Basic Explanation) 8:40-15:45
  - OOP Ex: Sketch Pad Dr. Ivan Sutherland ”Father of Computer Graphics 15:45-22:10 **Good for Classroom Instruction**
  - OOP in BYOB (Demo of Functions in BYOB) 22:35-29:20
  - Declarative Programming 29-22-31:20
  - Declarative Programming Examples in BYOB 31:25-35:20
  - Review of Paradigms 35:25-end
- BJC Lecture 6: Algorithms (With Luke Segars)
  - Computer Worms 0:00-1:30
  - Algorithm Concept Intro: Rubic Cube Competition 1:40-2:40
  - Algorithm Definition 3:20-4:20 **Good for Classroom Instruction**
  - Early Algorithms 4:25-5:55 **Good for Classroom Instruction**
  - Familiar Algorithms 6:00-7:30 **Good for Classroom Instruction**
  - Commonly Used Algorithms (Page Rank, etc.) 8:00-10:45
  - Choosing an Algorithm Technique 10:50-12:15
  - Ways to Tackle Problems (Brute Force, Top Down, Bottom Up) 12:20-15:30
  - Algorithms vs Functions and Procedures 15:30-16:00
  - Turing Completeness (Computer Theory-BYOB is Turing Complete) 16:05-21:15
  - Algorithm Correctness 21:25-26:00
  - Algorithm Summary 26:00-end
- BJC Lecture 7: Algorithm Complexity
  - Yahoo predicts America’s Political Winner 0:00-1:25
  - Function Abstraction (Explanation of Functions and Algorithms) 1:28-2:45
  - What is IN a Spec 2:45-3:30
  - What is NOT in a Spec 3:30-5:15
  - Reference Text “Introduction to Algorithms” 5:18
  - Algorithm Analysis: The Basics 6:00-7:40 ** Good for Classroom Instruction**
  - Algorithm Analysis: Running Time 7:41-8:25 **Good for Classroom Instruction**
  - Algorithm Analysis: Runtime Analysis Problem and Solution 8:25-9:55
  - Runtime Analysis: Input Size and Efficiency 9:58-11:25
  - Runtime Analysis: Worst of Avg Case 11:25-13:20
  - Run Time: Final Assessment 13:20-16:46
  - Example:Finding a student by ID (detailed explanation of input/output) 17:00-31:20
  - Ex: Finding a shared birthday 31:21-33:30
  - Ex: Finding Subsets 33:40 to End
- BJC Lecture 8: Concurrency (Yaniv Assaf)

  - Friendship Paradox: Facebook 00-1:30
  - Concurrency & Parallelism (Inter-Intra Computer, Cloud Computing) 1:31-4:10
  - Anatomy of a Computer (John von Neumann Architecture) 4:15-5:20**Good for Classroom Instruction**
  - But what is INSIDE of a Processor 5:20-6:30
  - Moore's Law Prediction (2x Transistors/chip every 2 years) 6:35-7:45
  - Moore's Law & Related Curves 7:50-10:00
  - Power Density Prediction circa 2000 (Heat as an issue) 10:00-11:40
  - Multiple Core and Energy Efficiency 11:45-14:40
  - Energy & Power Considerations 14:45-15:40
  - Parallelism Again (What's different this time?) 15:41-16:40
  - Speedup Issues: Amdahl's Law 16:42-19:50
  - Background:Threads (Threads of Execution" is a single stream of instruction) 19:55-21:15
  - Speedup Issues:Overhead 21:15-23:50
  - Parallel Programing Example in SNAP BYOB (Race Condition) 23:53-26:50
  - Another Concurrency Problem (Deadlock and Livelock) 26:55-29:30
  - Summary "Sea Change" of Computing 29:30 to End

- BJC Lecture 9: Recusion
  - Movie "Inception" as an example of recursion 0:00-0:50
  - Recursion 0:50-1:40
  - Recursion Demo in Snap 1:40-17:00
  - Overview 17:00-21:00
  - Definition of Recursion 21:00-24:30
  - Examples of Recursion (You Already Know It!) 24:30-26:20
  - Trust the Recursion 26:22-29:40
  - Summary of Recursion 29:40-End

- BJC Lecture 10: Social Implications of Computing
  - META: Computers in Education (Implications of Multiple Choice Tests) 0:00-4:30
  - Computers in Education (Open?) Judah Schwartz 4:31 –
  - `Tools 4:50-5:30`
  - `Microworld 5:30-6:30`
  - Microworld Example Physics Simulation 6:30-10:30
  - Courseware 10:38-11:30
  - RSA Animate: Changing Educational Paradigms 11:35
  - Animation Begins (Sir Ken Robinson: Changing Paradigms) 12:30-24:25

- BJC Lecture 11: Recursion II Alijia Yan
- Mobile World Congress 0:00-2:15
- Recursion:Factorials (Factorial (n)+ n! 2:30-7:40
- Fibonacci and Fibonacci Series Video 7:45-11:45 **Good for Classroom Instruction**
- Fibonacci Ex: fin(n) Math and SNAP blocks 11:50-13:15
- Example of Recursion: Counting Change 13:20-17:30
- Call Tree for "Counting Change" with SNAP example 17:35-22:50
- Summary of Recursion 25:40-26:21
- BJC Lecture 12: Social Implications II Dr. Gerald Friedland **Good for Classroom Instruction-Suggest Previewing due to Social Media Examples.** (This would be a good suppport for Social Media Safety/Awareness lesson)

- Dr. Gerald Friedland Sr. Research Scientist at International Computer Science Institute (ICSI) on Sharing Multimedia and the Impact on Online Privacy) 0:00-1:45
- Introduction to Social Media: The Price of Social Media Use-Stephen Colbert 1:50-6:25
- Observations on Sharing Data and Ineffective Privacy Protection 6:30-7:50
- Social Cause: Collection of Data Across Sites 7:50-10:30
- Multimedia in Internet is Growing 10:35-12:05
- CS Problem: Higher Demand for Retrieval and Organization of Data 12:07-13:05
- Manual Tagging & Geo Tagging 13:05-17:30
- Issue of Tracking & Dangers of Oversharing 17:30-18:31
- Berkeley Multimedia Location Estimation Project 18:31-28:14
  - ICSI's Evaluation Results 19:49
  - YouTube Cybercasing 20:47
  - Privacy Implication of Internet and Data 22: 30
  - Person Linking Using Internet Videos 25:45-26:45
  - Solutions for Privacy that Don't Work: Think Before You Post! 26:45-28:14
- BJC Lecture 13 is Not Available
- BJC Lecture 14: Human-Computer Interaction Bjorn Hartman
- Bjorn Hartman Background 0:00-3:30
- Human Computer Interface(HCI) 3:45-6:00
- HCI: Design, Computer Science, Applied Psychology 6:00-8:00
- Iterative Design Cycle 8:00-10:30
- Understanding Users 10:35-11:35
- Prototype Interface Examples 11:40-14:00
- Evaluation (Formative, Summative) 14:50
- Why Study User Interfaces**Good for Classroom Instruction** Ex:Mouse Xy axis, Sketchpad, PC, Tablets 15:00-25:00
- What had changed? Research: Mainframe to Ubiquitous Computing 25:00-29:30
- Example Project: Using Dexterity for Computer Interface Video 28:30-29:30 **Good for Classroom Instruction**
- Zipf/Power Law Distribution 30:00-32:00
- HCI Research at Berkeley 32:10-46:25
- Multi Touch Apps and Toolkits **Useful for Classroom Instruction**
- BJC Lecture 15:) Artificial Intelligence- Anna Rafferty -Anna Rafferty 0:00-1:00
- Definition of AI 1:00-1:48
- John McCarthy AI definition 1:50-2:30
- AI History and Explanation 2:35-6:40
- Revival of AI: Rules & Concepts 6:45-10:20
- AI and Intelligence (What intelligent things do people do?) 10:25-11:52
- Tour of AI applications 11:55-12:30
- AI Planning 12:30-14:50

- Machine Learning 14:50-18:58
- Robot Learning to Walk (Video) 18:58-20:25
- Natural Language Processing 20:30-23:15
- Unsupervised Learning Ex. 23:20-25:00
- Robotics 25:00-30:05
- Automatic Towel Folding Robot (video) 27:40-29:45
- Recap of AI 30:10-31:15
- Turing Test of Intell 31:15-34:15
- Summary 34:20-35:53
- BJC Lecture 16:) Computational Game Theory
- Checkers (Weakly) Solved 0:00-1:17
- Computer Science Game Theory 1:20-1:35
- CS- A USV View 1:36-2:16
- The Turk (1770) 2:20-4:00
- Claude Shannon 1950 "Father of Informational Technology) 4:05-5:10
- Deep Blue vs Gary Kasparov 1997
- What is "Game Theory" 11:41-12:40
- What "Board Game" do you mean? 12:41-13:25
- What is a "Strong" Soluntion 13:28-15:00
- Game Crafters (Strongly Solve) 15:11-20:15
- Strongly Solve Ex. Video "War Games"(1983)16:26-19:41
- Weakly Solving a Game (Checkers) 20:20-22:12
- Strongly Solving Ex:1,2:...12) 22:20-35:00
- Ex: Tic Tac Toe 27:25
- Demo 29:20-33:15
- Games Crafters Revisited 35:00-35:50
- Future Application 35:51-36:47
- BJC Lecture 17:) Higher Order Functions
- Coding is Cool Again 0:00-0:45
- High Order Function Introduction 0:47-8:50
- Higher Order Function Demo in Snap! BYOB (Functions and Blocks are commented out) 8:51-37:35
- BJC Lecture 18:) Distributed Computing
- 0:00-2:25 Super Computers Faster than 50M Laptops
- Lecture Overview 2:25-3:00
- Memory Hierarchy 3:00-6:00
- Memory Hierarchy Details 6:00-10:40
- Networking Basics 10:45-12:45
- Networking Facts & Benefits 12:45-13:30

- Performance Needed for Big Problems 13:30-16:40
- What can we do? Use many CPUs 16:41-18:05
- Distributed Computing Themes 18:05-21:20
- Distributed Computing Challenges 21:25-25:40
- Review: Map Reduce 25:41-26:30
- Google's Map Reduce Simplified 26:30-40:52
- BJC Lecture 19: Higher Order Functions II
- Busting the King's Gambit (Strongly Solve-Chess) 0:00-2:30
- Higher Order Function Review (Filter, Map, Reduce) 2:30-6:30
- Snap! BYOB Demo (Commented Out) 6:31-28:19

Standard Lab Day Lesson

Learning Objectives

Students will be able to...

- Describe difficulties they were having with the assignment on previous days
- Explain the requirements of the project on which they are working
- List questions they have about the current project

Materials/Preparation

- Current project specification
- Examples of necessary concepts or techniques
- Prior to each lab day, have a sense of students' difficulties with the current project so you will know what to address in the review portion of the lesson
  - Observe student progress throughout lab time and note common issues or misconceptions
  - Be aware of students who are more behind than most of the class
  - The exit ticket (see below) will help with gathering this information

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10-15 minutes	Review of trouble spots, concepts, misconceptions, etc.
30-35 minutes	In-class lab time
5 minutes	Exit ticket

Instructor's Notes

1. Review
   - Discuss and review areas in which a number of students are having trouble (or are expected to have trouble) with the current project
     - Using previous days exit tickets, questions from students, instructor awareness of trouble points in the project, and/or any other resources to determine what needs covering
     - Avoid lecturing or directly giving solutions; instead, present the problem and ask leading questions to help guide students to the solution
       - e.g. (from Project 1: "It looks like a lot of us are having difficulty making all our sprites move

        at the same time that the words change. Is there something we learned about that can trigger scripts in a bunch of sprites all at once?"
    - If some students appear to be further along, utilize them to help explain the tricky concepts to their classmates
  - Use this time as an opportunity to remind students of upcoming checkpoints, recent labs or activities that may be applicable to the project, and/or how far along they should be by the end of the day

2. Lab time
   - Allow students to work on the current project at their own pace
   - Provide a mechanism for students to ask questions of course staff as needed
     - Simply having students raise hands often does not work well, as it can be hard to keep track of in what order hands were raised; consider a queue of some kind where students write their names when they have a question
   - When there are no current questions, circulate and observe progress, stepping in if students appear stuck but are not asking for help
   - Be sure to meerkat and not spend more than a minute or two with any single student at a time

3. Exit ticket
   - Before students leave, have them answer the following questions on a small piece of paper:
     1. What was the last thing you accomplished on the project today?
     2. What is the first thing you will work on tomorrow?
     3. Do you have any questions about the project you would like answered? If so, what are they?
     4. Are there any topics or concepts that would help with the project that you would like reviewed? If so, what are they?
   - These answers will help you plan your review time for the next lab day

Accommodations/Differentiation

- Students who are consistently ahead and finish early can be encouraged to add extra features to their project for possible extra credit.
  - Beware of students who claim to be ahead or finished but are actually trying to avoid work. When in doubt, ask a student to demonstrate their project.
- Students who are consistently behind should get extra attention during lab time. If they continue to be behind, try to find a before- or after-school opportunity to provide extra assistance.
  - Students who are struggling significantly can be offered late turn-in, reduced requirements, or other "deals." **All students should submit work for ALL projects, even if it has to be simplifed or late.**

Culture Day Lesson A: Video/Reading

Learning Objectives

Students will be able to...

- Describe the key topics discussed in the video/reading
- Answer critical thinking questions about the video/reading
- Connect the video/reading to some aspect of their culture, society, or life

Materials/Preparation

- The reading or video for the lesson
- Critical thinking and/or analysis questions relating to the topic of the lesson

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
10 minutes	Introduction to topic and video/reading
15 minutes	Watch video/Read material
20 minutes	Class discussion or activity
5 minutes	Debrief and wrap-up

Instructor's Notes

Introduction to topic

- Provide motivation or context for the day's topics by asking questions connecting the topic to students' experiences
  - e.g. "How many play video games at least 10 hours a week? Do you get anything out of playing besides entertainment? Today we're going to watch a video about the positive benefits of playing video games."
  - This can be very brief to simply set up the video/reading, or it can be a deeper conversation to encourage students to consider their own positions on the subject. If taking the latter approach, some time will likely need to be moved from the "class discussion" section to this section
- Describe the topic of the video/reading and provide context on the presenter/author and the background on the specific video/reading.
  - Avoid getting into too much detail here, and *especially* avoid summarizing the material. Your goal is to provide context and background, not to preview the material itself.
- Present the critical and/or analytical questions that will guide the class discussion or activity later. Provide any necessary explanation or elaboration to ensure students understand the expectations.
  - In some cases, you will want to elaborate on the questions to get students all on the same page for the discussion or activity later. In other cases, you will want to be deliberately vague to encourage students to form their own ideas or interpretations.

Video/Reading

- Students watch the video/read the reading
  - Remind students to consider the guiding questions when watching/reading
  - Students may take notes if desired, but should give their full attention to the material
  - The room should be mostly silent for this part of the lesson, and students should not have access to computers
    - Some students may want to take notes on the computer, but doing so presents temptation to do other things. Paper notetaking is recommended.

Discussion/Activity

- Lead a class discussion or activity about the topics covered in the video/reading and guiding by the questions presented before the material. This can take one of several forms, including, but not limited to:
  - an open, full-class discussion
    - when using the approach, be sure that all students have a chance to contribute and that the conversation is not dominated by a few voices
  - small group discussions, after which one member of each group shares with the class
    - this can either be open-ended, allowing each group to discuss whatever they choose, or a "jigsaw"-style activity where each group is given one question to focus on
  - a "think-pair-share" activity
  - a structured activity to simulate or recreate something discussed in the material to allow students to gain a deeper appreciation of the topic

  - ∗ as an example, after a reading or video on computer security, students could play (or at least read about and consider) the game _Control-Alt-Hack

Debrief

- Ask one or more students to summarize the topics covered in the lesson and their thought or opinions
- Consider collecting some evidence of the activity, such as students' responses to the guiding questions or notes from small-group discussions, to evaluate engagement with the lesson

BJC Lecture Suggestion

Good for Classroom Instruction/Background Information for Instructors

- BJC Lecture 10: Social Implications of Computing
- META: Computers in Education (Implications of Multiple Choice Tests) 0:00-4:30
- Computers in Education (Open?) Judah Schwartz 4:31 –
- Tools 4:50-5:30
- Microworld 5:30-6:30
- Microworld Example Physics Simulation 6:30-10:30
- Courseware 10:38-11:30
- RSA Animate: Changing Educational Paradigms 11:35
- Animation Begins (Sir Ken Robinson: Changing Paradigms) 12:30-24:25

BJC Lecture 12:Social Implications II Dr. Gerald Friedland *Good for Classroom Instruction-Suggest Previewing due to Social Media Examples

- Dr. Gerald Friedland Sr. Research Scientist at International Computer Science Institute (ICSI) on Sharing Multimedia and the Impact on Online Privacy) 0:00-1:45
- Introduction to Social Media: The Price of Social Media Use-Stephen Colbert 1:50-6:25
- Observations on Sharing Data and Ineffective Privacy Protection 6:30-7:50
- Social Cause: Collection of Data Across Sites 7:50-10:30
- Multimedia in Internet is Growing 10:35-12:05
- CS Problem: Higher Demand for Retrieval and Organization of Data 12:07-13:05
- Manual Tagging & Geo Tagging 13:05-17:30
- Issue of Tracking & Dangers of Oversharing 17:30-18:31
- Berkeley Multimedia Location Estimation Project 18:31—
- ICSI's Evaluation Results 19:49
- YouTube Cybercasting 20:47
- Privacy Implication of Internet and Data 22: 30-25:40
- Person Linking Using Internet Videos 25:45-26:45
- Solutions for Privacy that Don't Work: Think Before You Post! 26:45-28:14

Accommodation/Differentiation

- Be aware of unconscious assumptions of context when presenting lessons relating to societal or culture topics. This is especially important in highly multi-cultural classrooms. When in doubt, ask questions or give brief explanations of any terminology, concepts, or ideas that are needed to appreciate the material.
  - For example, if showing "Smartest Machine on Earth", be aware that some students may not be familiar with *Jeopardy!*.
- Feel free to adjust the pacing guide liberally to meet the needs of your chosen material. If a video or reading will require more than 20-25 minutes, consider splitting the lesson across two days.

  - Day one should include introduction of the topic, the video/reading, and a brief reflection, with the discussion or activity pushed to day two.

- Try to vary the topics of culture days throughout the semester to engage a broad range of students' interests and experiences. Not all students will connect with every lesson, but you should strive to have every student connect with at least one or two culture days each semester.

Culture Day Lesson B: Student Research Project/Presentation

Learning Objectives

Students will be able to…

- Describe in detail the topic of their assigned/chosen computer science related topic
- Answer questions about their topic
- Explore and analyze the impact on, and impact of, technology in the context of their topic

Emphasize with students

Curriculum Competencies - Applied Technologies Coding is simply a technical skill. Yet the process of software development, product creation, and deployment, involves the intersection of diverse technical innovations, global community of peoples, cultural beliefs, values, and ethical positions. There is impact from, and impact on, our life and society at many levels (personal, community, global, and environmental), including the unintended negative consequences of the technology choices we make. These connections and discussions, help us appreciate the past, be wiser in the present, and more ready to embrace the future.

Possible Topics

- Famous figures in computer science (Donald Knuth, Alan Turing, Kernighan & Ritchie, Mark Zuckerberg, Bill Gates, Elon Musk, Steve Jobs, etc.)
- Famous women figures in computer science (Grace Hopper, Bletchley code breakers, Ada Lovelace, Dorothy Vaughan, Anita Borg, etc).
- Important technologies or algorithms (RSA, Dijstra's Algorithm, RAID, integrated circuits, etc.)
- New and emerging technologies (AI, Machine Learning, robotics, cryptocurrencies, etc.)
- Impact of technology on society (social media, health and lifestyle, screen time, etc.)
- Ethics (privacy, cyberbullying, security, etc.) This list should be expanded and updated, from time to time, for currency.

Materials/Preparation

- A list of possible topics for research projects
- Encourage students to research from online and other resources, and keep track of sources
- Citation generator http://www.easybib.com/k this is a handy way to generate citations (even for websites) that can be included as a Bibliography or References for the project
- Guidelines for projects and/or presentations
- Presentation Tips https://www.thinkoutsidetheslide.com/top-5-powerpoint-tips-for-student-presentations-in-school/
- Ideas for giving interesting presentations https://www.powtoon.com/blog/17-killer-presentations-tips-students-stand/

Pacing Guide

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Presentation #1
15 minutes	Presentation #2
15 minutes	Presentation #3

Duration	Description
5 minutes	Debrief and wrap-up

Instructor's Notes

Prior to Culture Day__

- Assign each student one or more topics to research and present to the class on a future day
  - Topics can be assigned, chosen by students from a pre-defined list, or suggested by students and approved by instructors
- Create a schedule of when culture days will occur and which students will present on each day
- Depending on how many students are in the class, and how many days you wish to allot for presentations, your pacing guide can be adjusted.

Student presentations

- Each student should give a 5-7 minute presentation on their assigned topic, followed by 8-10 minutes for Q&A
  - Students should have a visual aspect to their project (poster, PowerPoint, prop bag, etc.) as well as giving a verbal presentation
- Use your judgement regarding the level of technical detail expected in the presentation. It is probably not realistic to expect students to become experts in advanced technologies such as RSA, but they should be able to explain, at least at a high level, the details of their topic.
  - Do not allow students to simply read a textbook or online definition of the topic– ensure they can at least explain the subject in their own words.
- Allow classmates to ask questions, but beware of students trying to stump each other.
- Have a few questions for each assigned topic prepared ahead of time for instructors to ask in case classmates do not have questions.

Accommodation/Differentiation

- In smaller classes, each student may be able to present twice in a single semester.
- For classes where students are less experienced with presentations, consider a "science fair"-style event where students produce a display that can be viewed by others to present their topic.

Culture Day Lesson C: My Skills and Interests Journal

Learning Objectives

Students will be able to...

- Document their technology learning journey
- Self-reflect on the joys and challenges of technology learning
- Appreciate software development as a process

Emphasize with students

Curriculum Competencies - Applied Skills require self-evaluation Students may come to the course with many stereotypes/myths, about what they can do, or can't do. Minority groups may experience greater hesitations and apprehension about technology. By keeping a learning journal, for self-evaluation and self-reflection, students develop confidence. They will notice what skills they already have, what they have gained over time, what they are capable of learning, what their personal interests might be, and what they would like to learn in the future.

It's ok not to know everything. There is always something new to learn. Just take one step at a time.

Pacing Guide

Have students do self-evaluation journaling at different stages during the course, such as:

- first week of course, after unit 2, after unit 4, and last week of course

On Journal Writing days, teachers can omit bell work, and provide students with 10 minutes to do their writing.

Duration	Description
2 minutes	Welcome, attendance, (omit bell work), announcements
10 minutes	Journal Writing Time
38 minutes	A shortened regular class
5 minutes	Debrief and wrap-up

Materials/Preparation

- Students should have individual notebooks, a registered blog space, or worksheet handout to journal with.
- Teachers can provide a list of guidance questions (see below), or a survey that contains these questions.

Instructor's Notes

1. Beginning of the course

- Introduce students to the idea of a Learning Journal
- Create a schedule of when, or how frequently, students are to write in their Journal

Self-evaluation and reflection questions

- Beginning of the course entry
- What tools and technologies have you used before?
- On a scale of 1-5, how confident are you with them (5 being super-confident)
- What do you look forward to learning in this course?
- What are your worries and concerns about this course?
- other thoughts
- Intermediate entries
- What new technical skills have you gained since last time? (list them out)
- What other skills have you gained? (creating a plan, drawing a graphic design, giving a presentation, etc.)
- What did you feel you could do better with, or spend more time mastering?
- Looking at the new skills you have gained, or practiced, how do you feel about it?
- What did you like best about the course so far?
- What other skills would you really like to learn?
- What do you not like the most about the course so far?
- other thoughts
- End of the course entry
  - If you were to give advice to a new student who will start the course, what would it be?
  - Was learning new skills easier, or harder than you thought? Explain.
  - What are your project or design interests?
  - Would kind of skills would you like to learn next?
  - What kind of future jobs might use these skills?
  - Other thoughts

Emphasize with students (continued)

Curriculum Competencies - Applied Design is a process Like learning any new skill, or language, it takes time to get the hang of things. In computer programming, it's very normal, for a section of code to not work the first time.

Here are a few tips to help you enjoy the journey:

- **Tip #1:** If something doesn't work, don't be discouraged! It's part of the process. In fact, even experienced programmers spend a lot of time "debugging" (finding logic errors) in their design and code. It's detective work. This is when you can practice critical thinking steps of breaking down the problem, isolating the error, analyzing the values of variables, step by step. It can be frustrating. But it's also really fun and rewarding, when something "works"! Savor those moments. Stay calm, and carry on.
- **Tip #2:** Don't be afraid to ask for help - from peers and teachers. In fact, professional developers frequently consult different focus groups online to ask questions, and share solutions. One popular developer community is Stack Overflow https://stackoverflow.com/

3. Share learning journey with peers

- At the end of the course, ask each students to share a few key points with the class. This could be done during a end-of-term fun or party day. They can refer to their final Entry page in their Journal.
- Celebrate each student's progress in their learning journey!

Continue to Emphasize with students

Curricular Competencies - Sharing - Debunking stereotypes and myths Let's celebrate your learning journey! Don't let stereotypes, myths, or negative messages affect your motivation to learn technology, whether it's coding, creative user-centered design, or project management, and everything else in between. Don't just be technology "consumers". Be equipped to create and build your own uses for technology. The essentials skills you learn in this course will empower you to use technology in meaningful, ethical ways, for a better future.

Accommodation/Differentiation

- This exercise could also be conducted in the form of a series of online surveys. The survey should allow for written responses, so that students are free to enter their own thoughts.

Culture Day Lesson D: Interview with People in Technology

Learning Objectives

Students will be able to...

- Identify the people in their family, or community, who have technology-related jobs
- Create interview questions to find out about their jobs
- Conduct an interview
- Identify the different types of roles and skills needed in the technology industry
- Identify technology skills needed in non-technology organizations
- Reflect on how technology could be part of their future careers

Emphasize with students

Curriculum Competencies - Applied Skills and Technology Students may come with stereotypes and myths about technology jobs.

There are many different types of roles and skills needed in the technology industry. Not everyone is a coder; and not everyone is coding all day long. There are artistic designers, music creators, script writers, project managers, marketing specialists, quality assurance testers, front-end designer, etc.

As students get to know people in their community, they may even discover individuals who are willing to be mentors for your classroom in the days ahead.

Materials/Preparation

- [General tips for interviewing others that might help: http://provisional.com/employers/employer-interviewing-tips

Pacing Guide

This lesson can be done over a period of 2 classes.

In the first class, students prepare for the interviews. This class could be combined with a Journal Writing day nicely (see Culture Day C). Or, it could be combined with a regular project day, where students focus on working on a specific project.

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
15 minutes	Introduce lesson and warmup activity
15 minutes	Prepare interview questions
10 minutes	Other activity (suggestion: Journal Writing)
5 minutes	Debrief and wrap-up

In the second class, students share and report on the interviews.

Duration	Description
5 minutes	Welcome, attendance, bell work, announcements
45 minutes	Each group presents their interview highlights and key reflections
5 minutes	Debrief and wrap-up

Instructor's Notes

1. Introduction to the lesson, and warmup activity

- Introduce the lesson to students
- Identifying local industry
  - Before students start thinking about who to interview, find out how much your students know about technology companies in your city or town. Spend some time to search online. Write on the board some examples of technology companies, or companies that have technology departments.
- Identifying people in your lives
  - Ask each student alone, or with partner, to brainstorm on a piece of paper 3-4 people who have technology-related jobs. Encourage students to think of a different variety of jobs.
  - Encourage students to think of people that they know personally.
- Share and decide
  - Find a partner, and share your list with each other.
  - Agree on 2 people from your combined list to interview together

2. Preparing for the interview

- In your pairs, prepare a list of interview questions for each person (questions may be different)
- Encourage students to think about the different people around them:

    - those who work in technology organizations
    - those who have technical positions in other types of organizations
    - could be friend, relative, or even someone that you can approach
- Encourage students to create a variety of questions, for example:
    - what does your typical work day look like?
    - what do you like, or not like, about your job?
    - what inspires you?

    - what are some challenges?
    - what educational background is needed to do your job?
    - what kind of technical skills are needed in your job?
    - what advice do you have for students?

3. Conducting the interview

- Each pair should make a plan of how to conduct the interview in the coming week.
- This should be done outside of class time.
- Remind students to be professional and respectful when interviewing
- Contact the person first to arrange a time and place to meet
- Be punctual; if working with a partner, remember to introduce all parties involved.
- Tell interviewee the purpose of the interview is for you (students) to learn more about technology related careers
- Ask politely for permission to share some of their answers (within the classroom)
- Avoid questions that may be awkward or too personal (like salary), and always thank the person for their time.
- Take some notes, while still listening and being attentive
- After the interview talk with each other about key points, and personal take-aways
    - What surprised you?
    - What was interesting?
    - What did you learn?
    - Did you feel inspired or get ideas about possibilities jobs you could do in the future?
- Together, write 1-2 paragraphs of reflection
- Together, prepare a few PowerPoint slides (or photos, brochures, or grab bag items) to capture your interview findings and reflections, which to use for sharing with peers.

4. Share with peers

- Ask each pair to share highlights, and key reflections, with the class
- This can be done in one class period. The presentation time per group should be adjusted so to accommodate all students, so that no one feels left out.
- As a guideline for a 5-6 minute presentation, they can prepare 2-3 slides for each person interviewed.
- Students can also play short video clips from their interview as well.

Accommodation/Differentiation

- Students who have little experience with interviews may practice interviewing one another in the class with a few simple questions.

- Some students for a variety of possible reasons, may struggle to think of others to interview. They may be encouraged to consider teacher or staff in the school as potential interviewees. Be sensitive to assist students who are new to the community, those not living with family, those who don't speak English at home, or have other situations that they may feel embarrassed about.
- Working with a partner may alleviate some awkwardness for students who really can't think of anyone to interview.
- Working with a partner may alleviate some possible anxiety related to a face-to-face interview.
- Motivated students could video the interview as well, and show clips during peer sharing time, with permission from the interviewee. The final sharing could be done in the format of a prepared video.