Snap Out of It!

Target Grade: Elementary/Middle School
Authors: Dana Yang and Nico Carballal
Semester: Fall 2019

Brief Overview

At first, code seems like a foreign language, something that will take years upon years to even partially understand. For many of these students (and even some of our mentors!), this will be their first time coding, but what we hope to show is that code can be written by anyone! We will demonstrate logical thinking through a demo, create an algorithm through an offline activity, and use block programming through actual laptops!

This lesson hopes to:

- Introduce fundamental ideas and concepts of computer science through demonstration and activity.
- Prepare students to pursue STEM careers that involve problem solving and logical thinking.
- Show that code can be written by anyone!

Teaching Goals

- Students will understand how computers read and listen to commands written by programmers.
    - **Computer**: a machine that receives and processes information to give output.
    - **Programming language**: a set of words and grammatical rules that are used by programmers to write instructions for computers to follow.
- Computers think literally and follow instructions written by the programmer.
- Students will have a better understanding of what it is like to be a programmer, rather than a consumer, of technology.
    - **Algorithm**: A series of steps taken to complete a task.
    - **Program**: An algorithm that humans write in code for a computer to read and follow. When we run a program, the computer follows the instructions and carries out a task.

    - **Pair Programming:** When two programmers use a single laptop to code, switching off between two roles: one does the physical typing while the other reviews the work and collaborates to solve the problem.
- Computer science is possible and anybody, regardless of age or background, can be a coder!

Careers and Applications

The most essential skill of computer science is problem solving, a tool that anybody can use in their day to day life. Even if you don't end up pursuing computer science, the subject has logical thinking, like math, that can benefit you in everyday life and decision making. Furthermore, computers are a fundamental part of our everyday lives, and a computer science education provides valuable knowledge when it comes to so many of these activities.

Career wise, computer science is present in many different fields. From software or web development to predictive business analytics, there are many ways to apply a computer science education. CS is clearly relevant within software-based companies; however, it is also applicable in medicine (such as the Human Genome Project) and in environmental protection (through monitoring and parsing through data). At the end of the day, computer science is here to stay as industries become more and more reliant on it, so it's great to introduce students early to the field!

Agenda

- Introduction
- Module 1: Computer Talk (10 min)
- Module 2: Code a Maze (10-15 min)
- Module 3: runFlappyBird() (30 min)
- Conclusion

Introduction

What's awesome about computer science is that it is easy to pursue on your own once you are introduced to it! However, many of the schools don't have access to the technology necessary to provide students with this type of education, and students may be intimidated by the idea of programming a computer, so be sure to introduce it in simple terms and with plenty of enthusiasm.

Explain to the students that computer science is the way we, as humans, can tell our computers to do what we want. The way we do this is through code, commands that we can write out and the computers can directly follow, step by step. Emphasize that computer science is logical and that code can be written by anybody!

Module 1: Computer Talk

Introduction

Students will learn how computers read and listen to commands written by programmers. Through an engaging demo, students will recognize that computers think literally and require specific instructions for optimal performance.

Teaching Goals

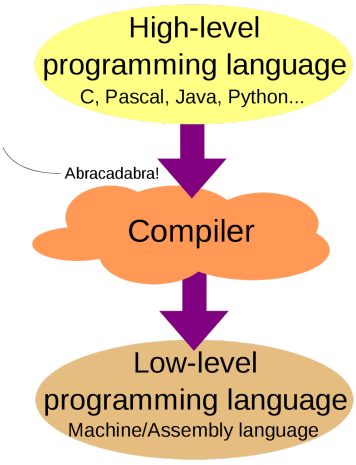
1. **Computer**: a machine that receives and processes information to give output.
2. **Programming language**: a set of words and grammatical rules that are used by programmers to write instructions for computers to follow.
3. Programmers write code using a programming language which the computer translates and executes.
4. Computers think literally, so it is important that our code is specific and exact.

Background for Mentors

Programmers write instructions, or **programs**, that computers read to perform various tasks. Programs are written in **programming languages**, of which there are two kinds: high-level and low-level programming languages. These languages are used to communicate with computers so that they can understand and follow our instructions. Computers are **literal**, so it is essential that our code is specific and exact.

Programmers use high-level programming languages like Python, Java, or Snap (Berkeley's block-based programming language!) to write programs. A *high-level programming language* is a set of English words and rules that follow special syntax and principles of logic. Syntax refers to the structure of words and symbols in a program, similar to the way grammar functions in day-to-day languages like English. High-level programming languages are human-readable and is independent of computer hardware.

Computers cannot understand words like humans do and are incapable of directly understanding programs written in high-level programming languages. Assembly languages and machine languages are *low-level programming languages* that the computer can directly understand. Low-level programming languages are written in a way that relates to the hardware of the computer. Assembly languages use mnemonic words and symbols that can be understood by humans. Machine language, on the other hand, use binary digits (0 and 1) and can only be deciphered by computers, not humans. A program called an assembler is used to translate programs written in an assembly language to machine code.



Any program written in a high-level programming language is called **source code**, which is human-readable. Any program written in machine language is called **machine code**. Machine code is code that the computer can directly understand and execute without translating. Source code is translated into machine code by a tool called the **compiler** for the computer to understand. Machine code is then executed by the central processing unit, which is a hardware device that executes the instructions of a program.

Just as we communicate with other people in languages like English or French, we can communicate with computers using programming languages. We, as programmers, write instructions for the computer to follow.

Materials

- Set of Markers per site
- 1 big piece of paper per site

Procedure

1. Tell students how programmers use programming languages like Python to write instructions for the computer to follow.
2. Have students give you step-by-step instructions for drawing an image. After each command, do as told. Have a mentor write down the spoken instructions on the board for the mentees to see.
    a. The options for painting are: a house, a car, and a flower. Give students these options, or any other appropriate drawing, to pick from!
    b. Have one mentor pin the painting paper on a hard surface or board while another mentor draws so that the students can see.
    c. The students' instructions should include taking the cap off, where to put the marker on the paper, and what colors to use!
    d. When following the students' instructions, make sure to take every command **literally** in order to demonstrate computers' literal thinking. One of the main goals of this module is to emphasize the importance of specificity in programming.
3. After the demo, go through the written instructions on the board with the students. Discuss which steps were important and which were not. Remind them that specificity is important when writing code.
4. Tell students that they have just created an algorithm!

Additional Notes for Mentors

Take every command literally! If an instruction is vague like "Take the marker and paint a line", you should take the marker, with the cap on, and draw a line. If students want you to draw a car, encourage them to give step by step instructions on how to draw each part of the car. Make this as playful as possible, and we encourage doing things like having a marker start on the floor!

Module 2: Code a Maze

Introduction

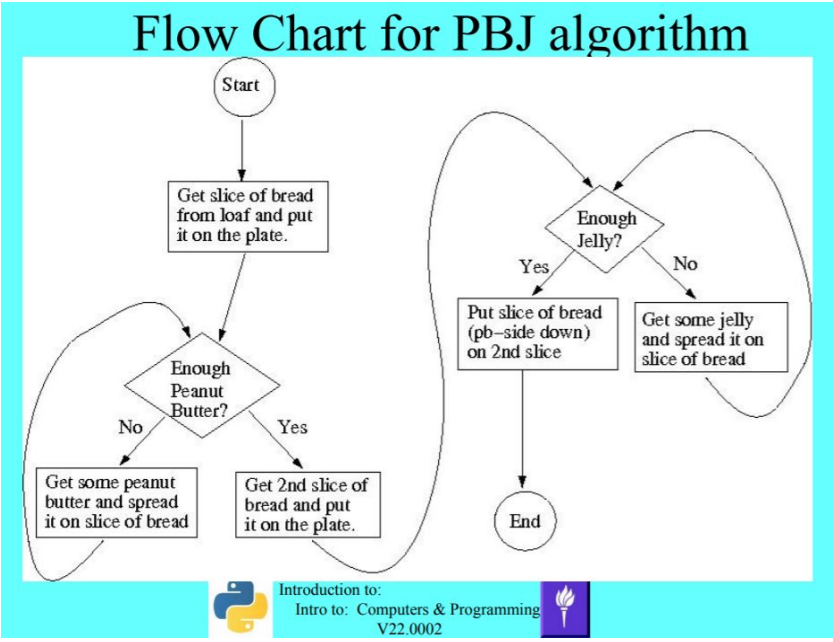
This module will introduce students to algorithms through an offline activity. Students will attempt to solve a maze by building an algorithm using pre-written blocks of code! This activity will familiarize the students with block-based programming before they work on actual laptops in the next module.

Teaching Goals

1. **Algorithm**: A series of steps given to a computer in order to complete an action
2. **Loops and If Statements**: Statements that allow your program to run a certain section of code for a specific number of times or for different conditions.
    a. **For loop:** Used to execute a section of code a specific number of times.
    b. **While loop:** Used to execute a section of code over and over until a certain condition is false.
    c. **If statement:** A coding statement that executes a section of code only when the condition is true.

Background for Mentors

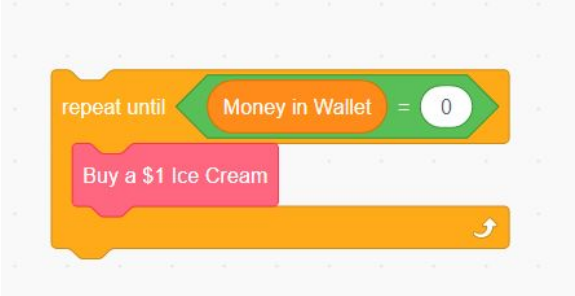
An **algorithm** is a set of instructions for a computer, or a person, to follow. Although the vocabulary will not have been introduced yet, students will have just created an algorithm in the last module by telling mentors step-by-step instructions on how to draw an object (such as a car). Another typical example of an algorithm is a recipe, since it tells the cook how to make a dish step-by-step. And of course, we also have mathematical algorithms, like how to find the mean of any set of numbers.



Algorithms are often repetitive, so it's necessary to use loops to ease the work on us as coders. Loops either repeat a certain section of code for a specified number of times or until a condition is met. For example, if I make $15/hour and work 8 hours in a day, there are two ways to write this: with and without a loop. Without the for loop (on the left) is lengthy and ugly; we manually write out how our boss pays us $15 every hour.

Instead, we used a **for loop**, which allows us to **repeat an action** (Boss pays me $15) **a given number of times**. In this case, the for loop essentially allows us to multiply our hourly wage by 8!

The **while loop** is used to repeat a section of code an **unknown number of times** until a specific condition is met. For example, if I want to buy as many $1 ice creams as I can, I want to keep buying ice cream **until I have no more money in my wallet**. The money in my wallet is my **condition**. **While** I have money in my wallet, I spend $1 on an ice cream. **Once** my wallet runs out of money, I stop buying ice cream.



Remember: the **for loop** is used to repeat a section of code a **specific number** of times. The **while loop** is used to repeat code an unknown amount of times **until some condition is true**.

There are **if statements** (conditionals), meaning that if a condition is specified, an action is performed. A condition can either be true or false. So, a simple example could be that **IF** you are hungry, eat a Snickers.

A more relevant example could be: **IF** it's raining outside, you should bring an umbrella. **ELSE**, you should leave your umbrella at home. The else part of a conditional is the part that is executed if the condition is NOT satisfied.

For this section, remind the students how this relates to the algorithm they created in module 1. For example, you could write an if statement: if the marker cap is on, take the cap off. Or, you could write a while loop: while the car has less than four wheels, draw another wheel. There's plenty of examples!

Materials

- Maze Blocks (Laser cut wooden blocks)
- Paper Maze
- Army soldier to dutifully follow our commands

Procedure

1. For this activity, students will attempt to complete a maze by putting together step-by-step instructions that lead them to the end of the maze.
    a. The blocks that they use are laser cut blocks modeled off of the Scratch block-based programming language. They should fit together nicely to create an algorithm.
2. Divide students into groups and have a friendly competition to see who can write a successful algorithm the fastest!
3. Maze:
    a. Have students create an algorithm using the instructions written on the blocks of code.
    b. For/while loops can be used to repeat a step multiple times. For loops are used when the number of steps is known. While loops are used when the number of steps is unknown. For example, a for loop is helpful when moving straight repeatedly for 3 steps.
        i. In Scratch, for loops are represented as "Repeat (# Times)" blocks and while loops are represented as "Repeat until (Condition)" blocks. The concept is the same; however, it's slightly easier to understand for users!
    c. Once they are done, have a mentor test their code by taking every step very literally. In this case, the mentor is the computer! Take a look at this link to see the maze and the solution!
        i. Maze Solutions

Additional Notes for Mentors

Any time you involve paper in a module, students may look the other way because they're so tired of seeing paper in the classroom. So, make sure to hype up the module and focus on the students working together. Divide the classroom into sections, and put students into groups. In addition, some of these algorithms are long and may be difficult for some students. **Make sure to help students along** in small groups; this will make everything much easier for the students!

Module 3: run FlappyBird()

Introduction

In this module, students will learn how to code using block-based programming languages on code.org! This activity builds off of the last module, in which students learned to write algorithms offline. Similarly, for this activity, students will write algorithms to complete different challenges on actual computers!

Teaching Goals

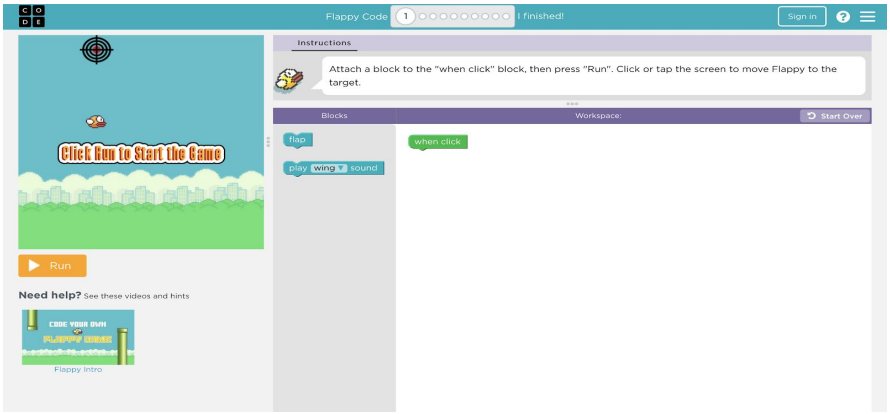
1. **Block-Based Programming Language**: A visual programming language like Snap or Scratch that uses and groups blocks of code to create fun games, projects, and animations.
2. **Program:** An algorithm that humans write in code for a computer to read and follow. When we run a program, the computer follows the instructions and carries out a task.
3. **Pair Programming:** When two programmers use a single laptop to code, switching off between two roles: one does the physical typing while the other reviews the work and collaborates to solve the problem

Background for Mentors

Designed for young learners and beginners, block-based programming languages like Snap or Scratch can be used to create engaging games, projects, and animations. Unlike text-based programming languages like Python, **block-based programming languages** enable users to drag and drop blocks of code while still using many important features like conditionals and loops.

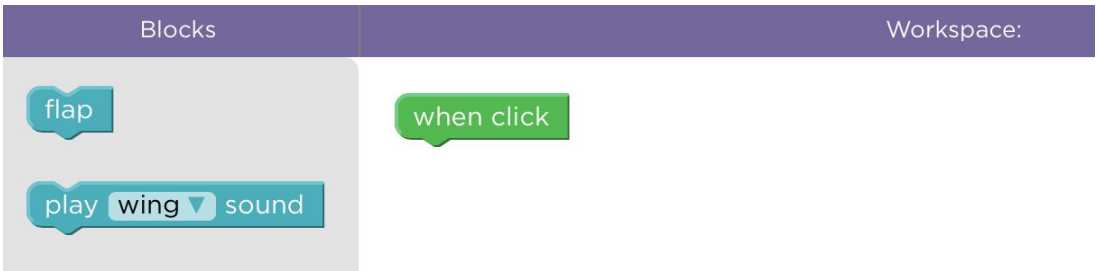
Below, we will explore Blockly on code.org in order to become more familiar with block-based programming. **Blockly** is a coding library that represents code as visual blocks that you can drag and drop. It is very similar to Scratch! To follow along, you can go to the link: https://studio.code.org/flappy/1 (this is the same link that the students will go to for the activity). For this module, students will learn how to create their own Flappy Bird game!

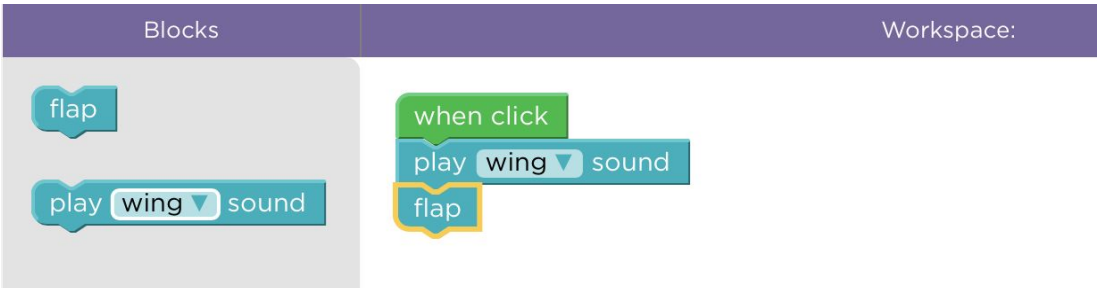
In the lesson, students are going to have the opportunity to work together to solve problems! While coding is often viewed as an individual activity, pair programming is a common practice to reduce inefficiencies and mistakes. **Pair programming** is when two programmers use a single workstation to code, switching off between roles. One does the physical coding on the laptop while the other helps review the work and thinks through the problem. For example, if one student is working on Hour of Code, the other student may ask probing questions such as, “Don’t we want to reset the score to 0 every time we crash into an obstacle?” Although backseat drivers are often annoying, in this case, it can really help solve the problem quicker!



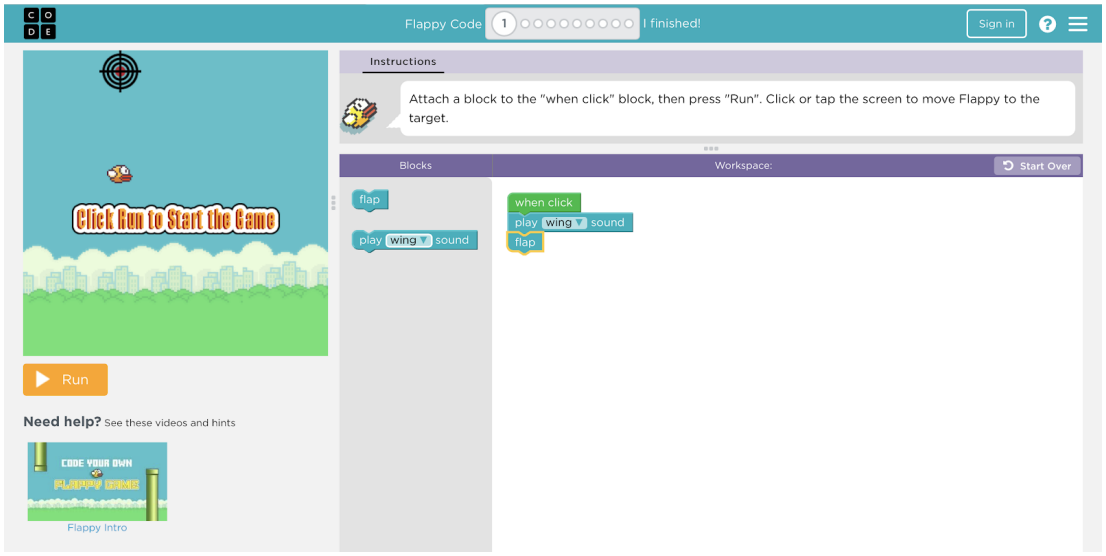
Blockly, like other block-based languages, provides many tools and areas for coding. The column labeled “Blocks” has all the code blocks that you can use to write your program. In order to create your program, you can drag and drop the blocks into the section “Workspace” to control how your game behaves.

The code blocks determine the motion, looks, sound, events, and other essential components of the game. Each block has a certain command and can fit with other blocks to control how your game runs. You can drag and assemble your blocks to begin writing your code.

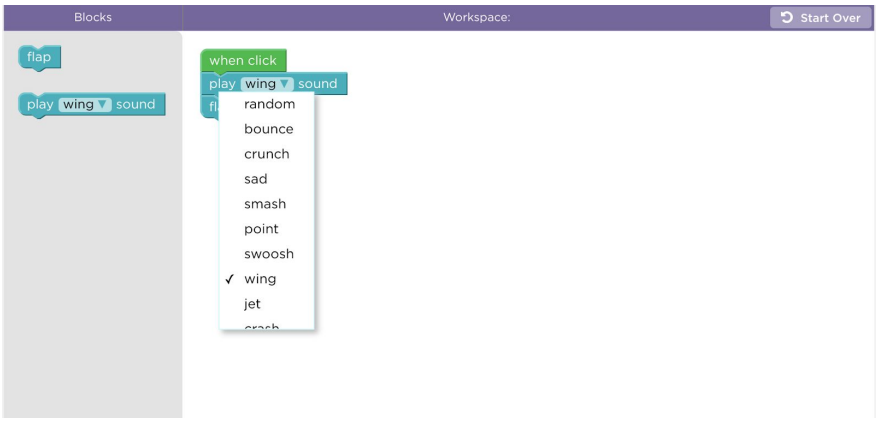




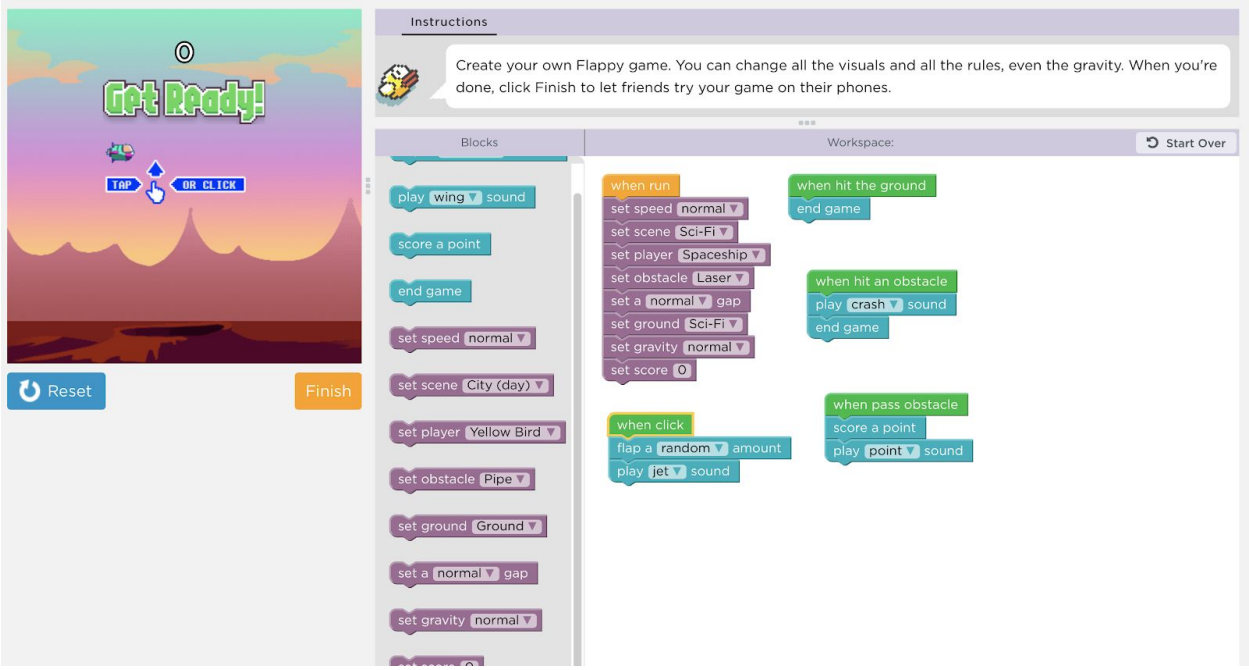
Once the blocks are assembled, you can click the orange “Run” button to run your program. After you run the program above, the bird should flap up and make a “wing” sound whenever you left click the mouse.



If you want, you can also change the sound the bird makes by clicking on the button with the drop down arrow.



There are many other features you can use so feel free to explore them as you go through the lesson plan!



Materials

- 1 Laptop per 2-3 students
- Laptop Chargers

Procedure

1. Pair students up and give each group a laptop.
2. Have students go onto https://studio.code.org/flappy/1.
3. Students should be brought to a page as described below
    a. A challenge is introduced in the instructions. In this case, the challenge is to move Flappy to the target
    b. The “Workspace” is where the student can place code that is going to be run!
    c. The “Blocks” section contains the possible blocks that the student can use to accomplish the goal. They will need to drag and drop blocks into the “Workspace” from the “Blocks” Section
    d. The “Run” command plays the code currently in the workspace. You can press “Reset” to go back and edit your code
    e. Make sure the students are practicing pair programming! If there are three students in the group, have one code and the other two be the backseat navigators.
4. If a group of students finishes early, there are other online activities they can do:
    a. For **all sites**:
        i. Hour of Code is primarily self-taught, meaning that students will be given the freedom to explore it at their own pace. Here’s another Hour of Code tutorials students can try:

1. Minecraft Hour of Code
    a. Link: https://code.org/minecraft
    b. Difficulty: Beginner (same level as Flappy Bird)

ii. Barebones Snap
    a. Link: https://snap.berkeley.edu/snap/snap.html
    b. This allows kids to freely explore Snap (talk about how this was developed in Berkeley!) to their own liking!

b. For **more advanced sites**:
    i. If the students find the activities above too easy or boring, have them try the Emoji Challenge, a project created by ANova.

1. Emoji Challenge: Have students write an algorithm that draws the emoji below using Berkeley's visual programming language, Snap. Have them use the motion and pen blocks to draw their image. For more specific instructions, look at slides 8-18 in Emoji Challenge - Anova Slides.

Additional Notes for Mentors

Being on a computer can distract any group of kids. Ensure that WiFi on the laptops is used responsibly and keep a close eye on the students. **Laptops and computers are expensive, so make sure they are not abused in any way.** Lastly, direct the students through the thinking process of creating an algorithm and make sure students are working together through pair programming!

Conclusion

Go around to students individually and ask what they were able to complete! Hopefully students will be amazed by the fact that they were able to make a Flappy Bird game or that they could even tell the computer to complete actions! Remind the students that they can do coding practice at home. Tell them to ask their parents first, but there are plenty of online tutorials and games for learning how to code.

Lastly, **a HUGE shoutout to Berkeley Anova** for lending us laptops for this lesson. Without their generosity, we would be unable to introduce computer science to all of our students!

Additional Notes for Mentor Development

Written by Cammie Young and Nicole Zhu

What should you do if a student is frustrated with a concept?

1. Redirect their attention to the **basics** of the lesson since computer science concepts can always be broken down into very basic steps. For example, when talking about while loops, you could readdress the step-by-step instructions of Module 1 and simply add the fact that while loops do the same thing, they have a specific goal that they want to reach.
2. Use **analogies**! For the for loops, you can simply compare it to a never ending rollercoaster, making it rounds around the course until someone gives it the instructions to stop.
3. Switch up the way both of you are approaching the code. Sometimes it's easier to draw things for one to understand! The lesson provides some fantastic diagrams (see flow diagrams for module 2) that you can adapt for other modules. Other times, it might help having your mentees verbally walk you through their code in order to see their thought process more clearly.

What if you, a mentor, are confused about a concept?

1. **Prepare by reading the lesson THOROUGHLY beforehand!** We know that in the midst of midterm season, the last thing you probably want to do is to read over a lesson plan, but doing so will help prepare you for the subject material - especially if you are unfamiliar with this week's computer science concepts. Doing this beforehand also gives you time to look up any of the teaching concepts. And definitely feel free to reach out to other members in BEAM if you have questions!
2. **Communicate with your site** beforehand to see who has experience with computer science. Distribute the teaching roles based on strengths and knowledge about the topics. Don't be afraid to take on a role even if you aren't completely familiar with it. Teaching the kids could help improve your understanding about the concept.
3. Accept the fact that it is okay to make mistakes when teaching these kids. In BEAM, we stress the importance of teaching these kids STEM concepts, but we also want you guys as mentors to **INSPIRE** these kids to pursue STEM careers. A few errors will not ruin STEM for these kids, but go into the classroom with the confidence and passion to teach these kids!

Final note for both mentees and mentors:

1. When in doubt about the veracity of an algorithm, walk through it step by step and try to break down what the computer is supposed to do versus what the algorithm is telling it to do. Try doing this in the smallest increments possible (ex: The player will move to the end of the maze hallway vs. the player will move a step to the right). This step-by-step process is very similar to **debugging** - a method programmers use to find errors in their code. If your mentees (or you, yourself) feel discouraged, remember that bug-finding can

be very frustrating/time-consuming for experienced programmers as well, so it's okay not to get things on the first try!

References

- Layers of Programming: Machine, Assembly, & High Level Languages, The Revisionist. https://therevisionist.org/software-engineering/cnss/programming-concepts/
- Coding a Lego Maze, Research Parent. https://researchparent.com/coding-a-lego-maze/
- Block-Based Coding, Scratch. https://en.scratch-wiki.info/wiki/Block-Based_Coding#Advantages_and_Disadvantages
- Hour of Code Activities, Code.org. https://code.org/learn
- Emoji Challenge - Anova Slides

Summary Materials Table

Material	**Amount per Group**	**Expected $$**	**Vendor (or online link)**
Markers	A set per site		Inventory
Paper	1 per site		Inventory
Maze	5-8 per site		Printed
A set of Code Blocks	5-8 per site		Jacobs Laser Cutter
Laptops			BEAM (Chromebooks) Berkeley Anova (MacBooks)
Laptop Chargers			BEAM (Chromebooks) Berkeley Anova (MacBooks)