HOPSCOTCH

## DEAR HOPSCOTCH TEACHER,

We are so happy that you have taken an interest in teaching your students computer science!

As you may already know, computer science is a discipline that is increasingly necessary for success in modern careers. By 2020, two-thirds of computing jobs will go unfilled because the United States produces too few computer science majors. 70% of new jobs in science will be in computer science. Future jobs in nearly every industry will require data analysis, and even fields like law already require paralegals to learn scripting to search for documents in large databases.

Beyond these practical applications, there are other great reasons to teach your students computer science!

## WHY TEACH COMPUTER SCIENCE?

Computer science (also known as computer programming, computer coding or computer engineering) is perhaps the most misunderstood discipline. Many inaccurately use the term "computer programming" to refer to the study of computer programs, like Microsoft Word or how to use Google effectively. Others see coding as a trade skill, much like plumbing or electrical work. However, computer science is neither. Rather, it is the study of computational thinking, or how to use logical thinking and abstraction to **develop generalized solutions to complex problems.**

Just as English is not about grammar, and history is not about memorizing dates, computer science is not really about code or computers.

Just as we ask students to make connections between events in history, we ask students to investigate the interactions between complex systems in computer science. When students write geometric proofs in math, we ask them to leave little room for ambiguity and analyze the sequence of steps thoroughly; we do the same in computer science. In English class we ask students to understand literal words as symbols and abstract entities in support of a larger theme, and in computer science we encourage students to take literal numbers and letters and endow them with complex meaning within a system.

Your students' **critical thinking skills will improve** by learning computer science, and it will show in their performance in other disciplines. We hope these activities engage and inspire your students. Enjoy!

--The Hopscotch Team

# HOPSCOTCH

TEACHER CURRICULUM

## LESSONS

1: What is Computer Science and Hopscotch?

2: Loops!

3: Loops continued!

4: Totally Random

5: Get Organized

6: Put It All Together!

7: Final Project

### GETTING STARTED

To aid in your endeavor to teach your students computer science, we are providing you simple lesson plans. Each plan is flexible: should it not fit your schedule or meet your age group's needs, it is perfectly acceptable to cut the plan short or combine lessons into longer sessions as needed.

Materials:

- 7 flexible, 45 minute lesson plans. We recommend planning for 8 just in case things move slowly. Target age is 6th-8th grade.
- Supporting slides
- The Hopscotch language and application (iPad only as of Sept 2014; download in the App Store)

**No experience necessary,** but we recommend you walk through the lessons beforehand.

LESSON 1

# What is Computer Science? and Introduction To Hopscotch

## Teacher Brief

The first and perhaps most important lesson of computer science is the notion that **computers do what they are told, and only what they are told, in the order they are told.**

If one fully understands this concept, and begins to think of everyday processes (making a sandwich, getting to school) as a set of instructions, they will begin to think like a computer scientist without trying all that hard!

This lesson focuses on identifying everyday things (phones, computers, etc.) that only work when given instructions, getting students comfortable writing instructions in Hopscotch, and ensuring that students understand that the order of instructions matters.

Hospcotch uses some computer programming vocabulary that you may want to review with students as you get started:

**Event:** An action that the computer recognizes and that causes it to do something is an event. In Hopscotch, events include "When the iPad is tapped" or "When the project is started"

**Abilities:** When you save a set of blocks, it's called an ability. We have some pre-made abilities for you to use when you first open Hopscotch, including "Dance", "Draw a Triangle", and "Go For a Walk". What we call abilities in Hopscotch are known as functions or subroutines in other programming languages. Easily replicable routines are a key concept in computer programming, and allow you to scale your code and create complex programs.

**Rules:** Rules tell your object what to do and when to do it. When you make an ability and pair it with an event, you create a rule. For example, if you choose the event "When stage is tapped" you can then pair it with the "Jump" ability. This creates a new rule: every time you tap the stage, your object will jump.

## Curricular Goals

- Basic understanding that computers only do what they are told, in the order they are told (step by step thinking).
- **Comfort with making mistakes and taking risks while writing code.**
- Age appropriate understanding of computer science and programming languages with real world examples:

  4th Grade: Students understand that computers can be programmed to solve problems. Examples include cell phones, computers, calculators.

  5th and 6th Grade: Students can identify more abstract uses of computing including GPS devices, cars, watches, etc.

  7th Grade: Students can identify potential problems that might be solved by computers (e.g. computers might one day be able to predict the spread of diseases)

  8th Grade: Students can easily spot uses of computing and potential problems that can be solved by computing. Point out problems that are more difficult to solve via computer than others (e.g. GPS vs. life advice).

SLIDES

5 min.

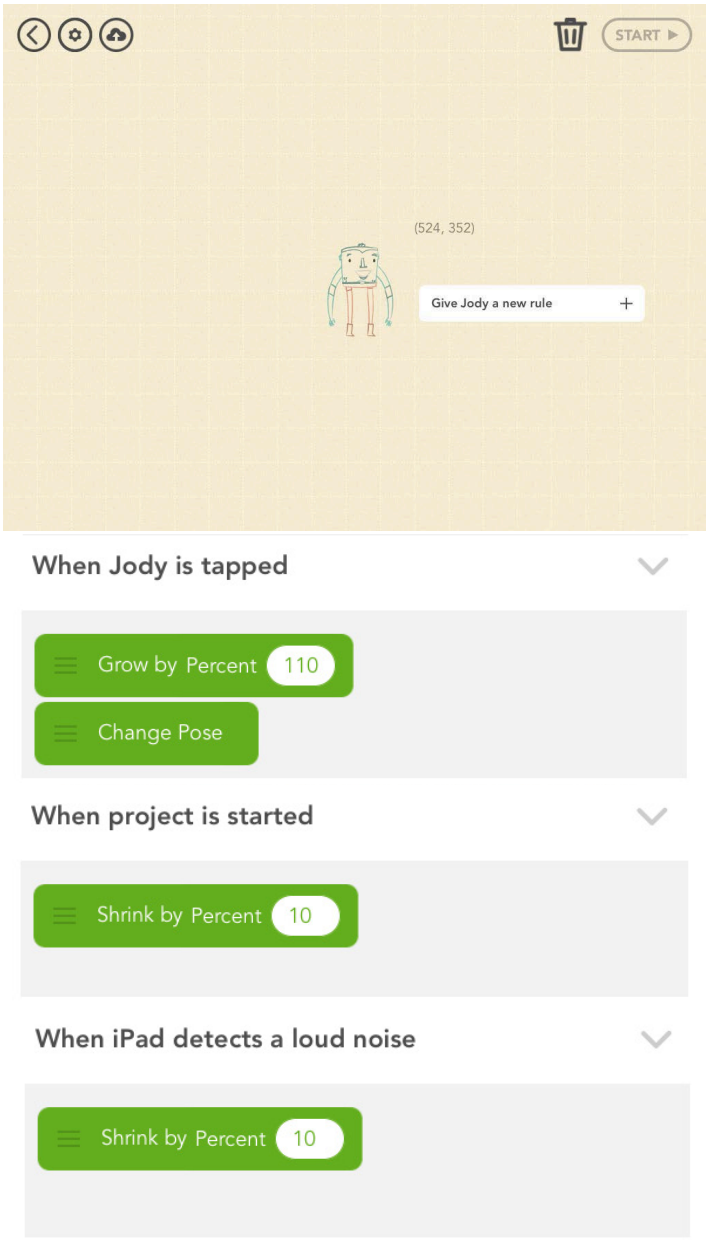
**Lesson 1 Slides: "What Is Computer Science?"** Use the associated slides to introduce the concept of computer science to your students.

CODE DEMO

5 min.

**Monster Hugs**

Show your students the Monster Hugs program. Then, rewrite it for your students to show them how it works. This is meant to be a simple but powerful example of how programming languages work.




Monster Hugs code

CODE DEMO

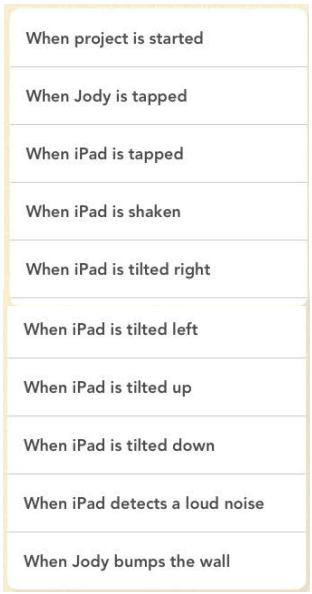
5 min.

## Menu Walkthrough

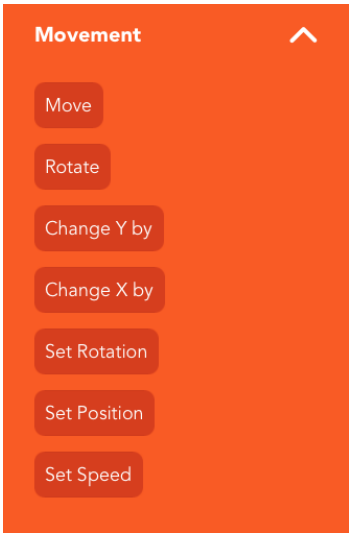
Create a new Hopscotch program and walk through each menu (Movement, Drawing, etc.) and the kinds of blocks they contain. Show the result of one block from each panel as examples to students.

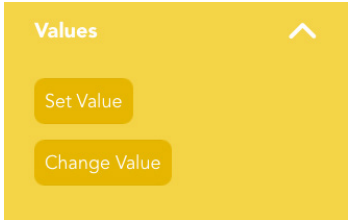
For example, use the move block to exemplify the Movement menu.

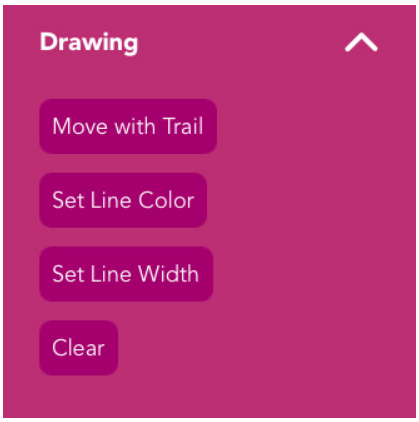
### Event Handlers

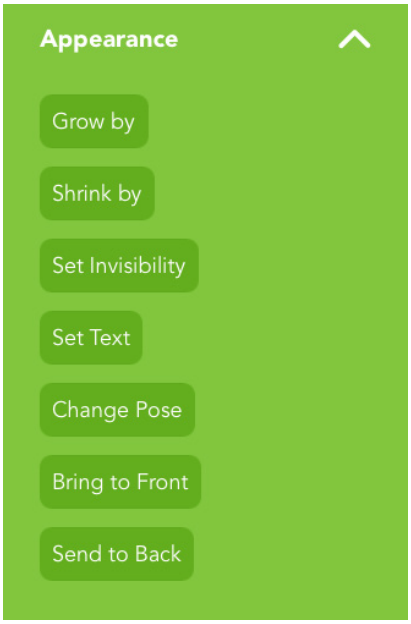


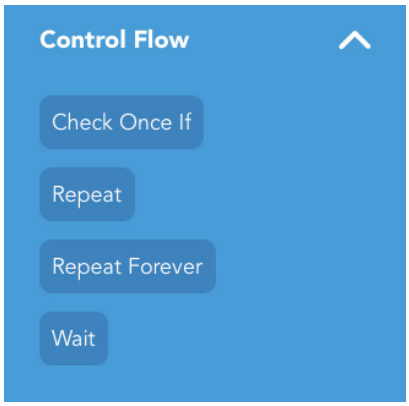

### Menu Blocks

10 min.

**Free Code**

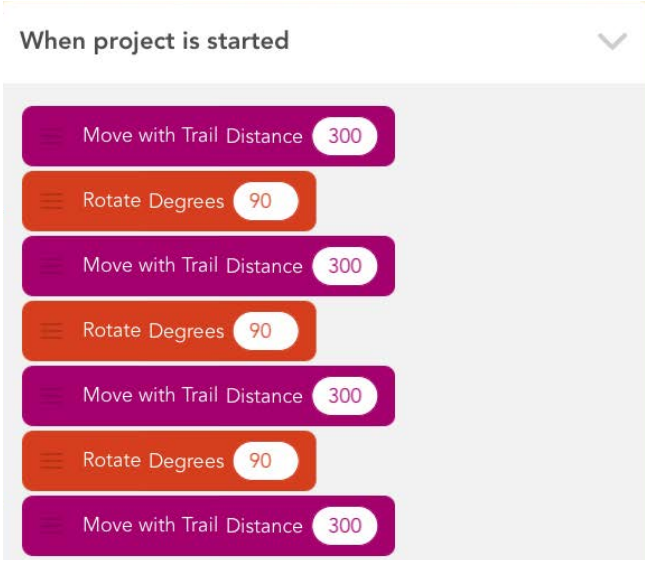
Students take 10 minutes to explore and try out the app. Should students be inhibited, have them start by having their object draw a line wherever it goes.

20 min.

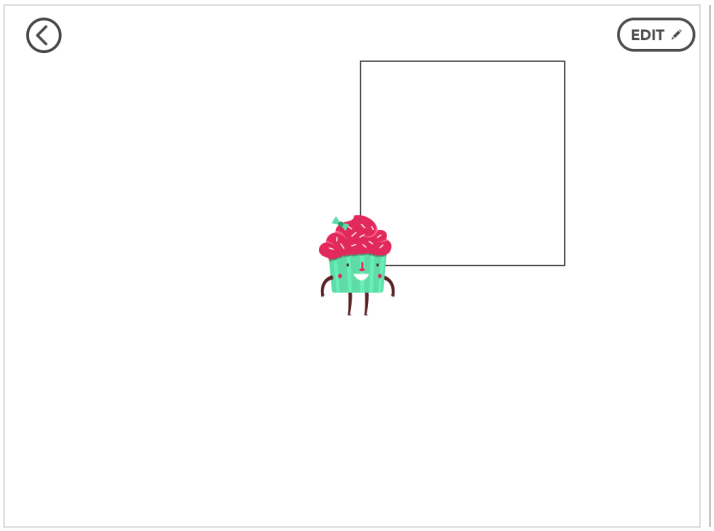
**Square Lab**

The first program we ask students to write is code to draw a square. We ask that this is done without using the repeat block so students can gain comfort in the environment with limited complications, and focus on sequential thinking.

- Begin by having students pretend they are the objects and walk in a square motion.
- Have students record the steps.
- Have them investigate which blocks are most like the steps they took.
- Begin coding.




Square Lab code




Square Lab project

LESSON 2

# Loops!

## Teacher Brief

As students have learned from the last lesson, computers have a finite set of tasks they can accomplish. But when these tasks are combined properly, amazing things can be built.

In addition to running instructions sequentially, computers are very good at **repeating sets of instructions.** In computer science we call this a **"loop"**.

Imagine that you are sending handwritten birthday invitations. While the names and addresses of your friends are different, the process of writing invitations is the same. You stuff the envelope, write the address, close the envelope, and add a stamp. You could use a loop to repeat this process: For the number of friends that I have: stuff an envelope, write that friend's address, close the envelope, and add a stamp.

There are actually many different kinds of loops. The two most popular are called **"for-loops"** and **"while-loops"**. For-loops repeat an instruction the number of times indicated by the programmer: "Clap once. Repeat **for** each student in the room." While-loops will continue to loop **while** some condition is true: "Clap once. Repeat while my hand is in the air."

This lesson is specifically about for-loops and getting students comfortable with using loop syntax. In Hopscotch, we currently only support for-loops. Our version of for-loops can be implemented by using the "repeat" block.

## Curricular Goals

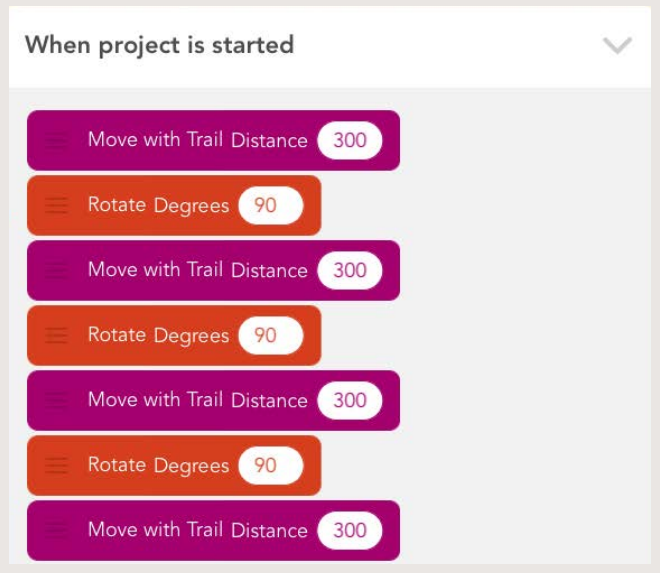
- Students gain more comfort and practice with sequential thinking.
- Students gain basic understanding of for-loop syntax.
- Basic introduction to abstraction: e.g. students can identify which lines of code represent lines vs. corners in a square.

CODE DEMO

5 min.

**Discuss Square Solution**

Walk through the solution code to previous lesson's square lab with students if further clarification is necessary. Ask students to tell you which commands represent which parts of the shape (ie: move commands are representative of sides, and rotations are representative of corners).




Square Lab code

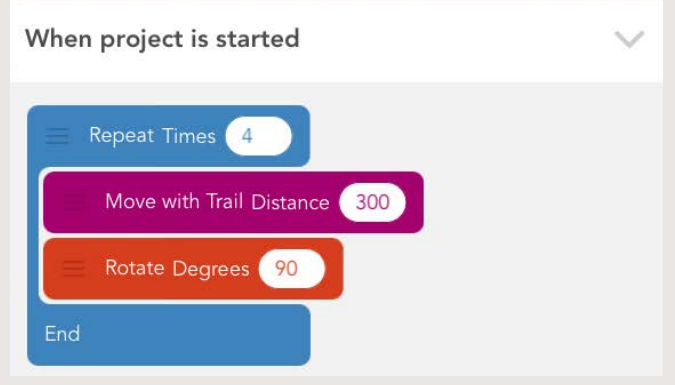
SLIDES

10 min.

**Lesson 2 Slides: "Loops!"**

Introduce students to loops using these slides.

Then walk through the square assignment using loops, as seen here, line by line.



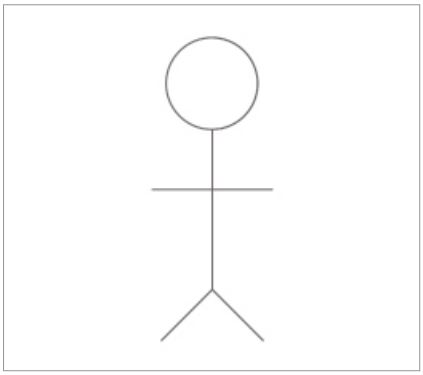

Square with loops

30 min.

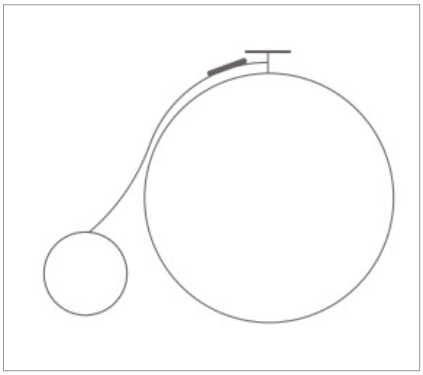
**Students use loops to draw a "Line Drawing" of their choice**

Show students examples of line drawings from the internet - drawings done using pen or pencil, and drawn in one continuous line, like this. After doing so, show them some of the examples of line drawings made in Hopscotch.

Star



Person



Bicyle

These are arranged in order of difficulty. The first is acceptable for 4-6th graders, the second for 7-8th, and the third could be attempted by advanced students.

**Food for thought: How do you make circles using loops?**

Give students the remaining class time to write code to create a line drawing of their choosing in Hopscotch. It is worth noting that drawings with curves are ambitious; have students choose subjects with few curves. Keep it simple! You only have 30 minutes.

LESSON 3

Loops Continued!

**Teacher Brief**

Looping is only powerful and useful to programs when the students can understand when and why it is used. Now that students are comfortable using loops in Hopscotch, they should begin thinking about loops in products and processes they use in their everyday lives.

Are there loops in the games that they play? For instance, in a soccer game, there are two halves and the exact same rules for a 45 minute game are repeated each time. Are there other loops that students perform in their everyday lives (e.g. eat cereal until it's done, run 5 laps in gym class)? Are there some loops that repeat for a definite amount of time and others that repeat indefinietly (e.g. stop light)?

Loops and sequential thinking are the building blocks of algorithms. The goal of this lesson is for students to understand the role of both of them, their interaction, and where each should be used.

**Curricular Goals**

- Students achieve basic proficiency in for-loops and understand when and why they are used.

- Students gain more comfort with very basic abstraction. Students should be able to identify loops vs. sequential instructions in apps they use or games they play, and identify why each was used.

  E.g. in Angry Birds, the game creators used a loop to make the birds at each level. The birds are identical from level to level. However, the creators could not use loops to build the different block structures that need to be destroyed at each level. Each block has a unique location. If loops were used, each level would be exactly the same as the last.

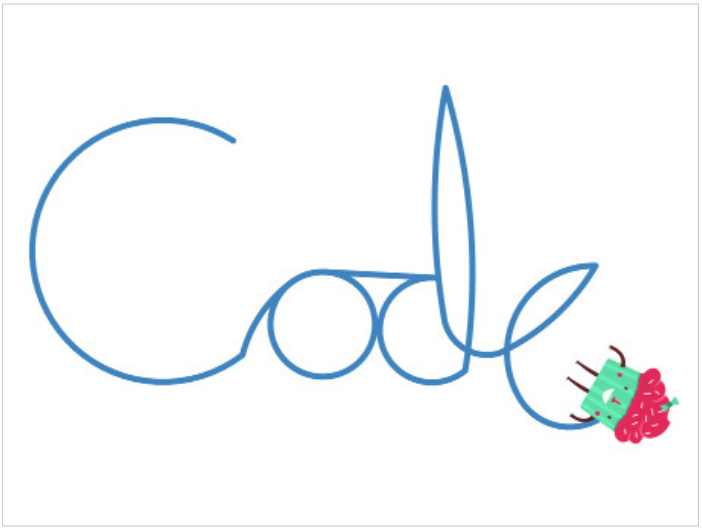
5 min. **Introduction To Robo-Signature**

During this class students will write code to draw their signatures in Hopscotch. Students should attempt to do so in cursive and not in block lettering. If this proves to be too difficult, or you think your students need more time practicing with loops, you may use block lettering instead.

40 min. **Robo-Signature**

Prior to beginning, have students write their name, initials or a word of their choice in cursive, paying close attention to where turns are made and where for-loops ("repeat" blocks) might be used to make curves. Once they have thought it through, have students work on their code for the remainder of class.

If a particular student or set of students are struggling with curves in loops, give them the smaller assignment of simply making a circle and help to walk them through this code. It's also worth noting that letters need not be perfectly round, smooth, or proportionate! It would take hours to get a perfect representation of your name in cursive in Hopscotch. Should you find a student is a perfectionist, be sure to gently push them to complete the assignment in the allotted time and perfect it later.



Robo-Signature

LESSON 4

Totally Random

**Teacher Brief**

In the last lesson, we learned that loops and sequential thinking are the building blocks of **algorithms**. Algorithms are at the heart of computer science: they are the recipes that computers follow to solve problems. Algorithms are most useful when they can be applied to a general problem. An algorithm to solve any Rubik's Cube is much more useful than an algorithm to solve just one possible Rubik's Cube. Often times the solution is simpler than you would think!

One way to help understand how an algorithm might be applied to a general problem rather than a specific problem is to imagine a maze.
One could solve this maze simply with a few commands:

Forward
Turn -90 degrees
Forward
Turn 90 degrees
Forward

However, if the maze is changed, this set of commands won't work anymore. The question becomes "how do we write an algorithm to solve any maze" rather than just this maze. Another example might be "how do we write an algorithm to find factorial of any number" rather than just one number.

Because the best algorithms are designed to deal with generalized and sometimes ambiguous inputs, the concept of randomness is very important to many algorithms. For example, if you claimed that your algorithm to find factorial of any number really worked, it could easily handle a slew of randomly generated numbers.

Randomness can also be used to make algorithms better. It is used in video games to simulate chance (e.g. a slot machine). It can also be used in our maze example. There is no set of specific commands that can solve any maze. However, if we continually try random commands (e.g. forward, turn by random amount) until we reach the end of the maze, we will eventually solve the maze.

One might ask, "why not simply follow the wall?" We must remember that computers are not like humans. They have no concept of what a wall is, or even the capacity to sense a wall. We'd have to write a new algorithm for that. However, there's no need to: we can get through a maze without worrying about walls at all. In fact, random movement is how the Roomba accomplishes the task of cleaning a whole room! This kind of randomness is found in nature as well. For instance, natural selection occurs when random gene mutations in an individual unexpectedly give it an edge over others in the species.

Algorithms and randomness go hand in hand conceptually. One represents the set of steps that solves a particular problem; the other represents the set of possible scenarios that it can be applied to.

We do not expect that students should master both these topics. Rather, students should begin to understand that **computers can solve general problems** and not just specific ones. Additionally students should feel comfortable using the "random" block and understand that using it will cause the outcome of their program to be different each time.

**Curricular Goals**

- Introduction to computational thinking and algorithms. Students should understand that algorithms are recipes that computers use to solve problems.
- Students should have a basic understanding that computers can solve general problems, not just specific ones.
- Understanding of how to use the "random" block.

## LESSON 4: Activities


10 min.

**Lesson 4 Slides: "Totally Random"**
Introduce your students to randomness in nature, science, and computer science with these slides. (The video on the last slide is optional, as content may not be appropriate to your age group).

15 min.

**Maze Kinesthetic Activity**
As a group, your class will attempt to construct an algorithm to solve any maze:

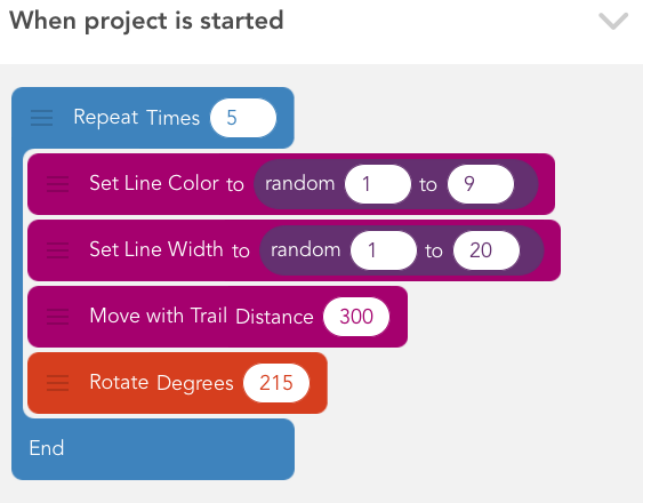
1. Begin by drawing a simple maze on the board and ask students to give you instructions to get out of the maze.
2. Ask students if this same solution would work if you changed the maze.
3. Ask students if randomness could be used to help someone get out of any maze while they were blindfolded. (e.g. move forward, turn a random direction -- given infinite time this will always work).
4. Have students kinesthetically try the solution. Select a goal point in the room (e.g. the door) that all the students have to get to. Then have students scatter across the room; distance from the goal does not matter.
5. Ask students to move forward by random amounts, and then turn by random amounts. You can select these numbers yourself, roll a die, or pick out of a hat. Play until most students have finished or time is up.
6. When complete, point out to students that distance from the door, or location, did not matter because all students eventually reached the goal through random movements. If some students did not reach the goal, explain that they eventually would have.

CODE DEMO

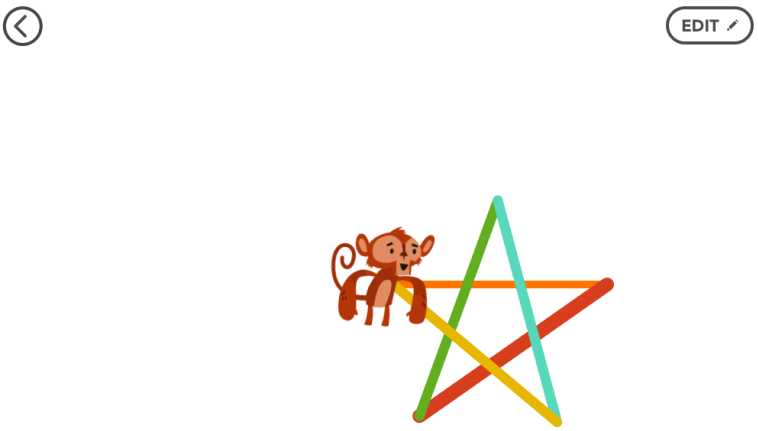
5 min.

**Randomness in Hopscotch**

Walk through line by line some of the examples of how randomness can be used in Hopscotch as provided here:




Random Code

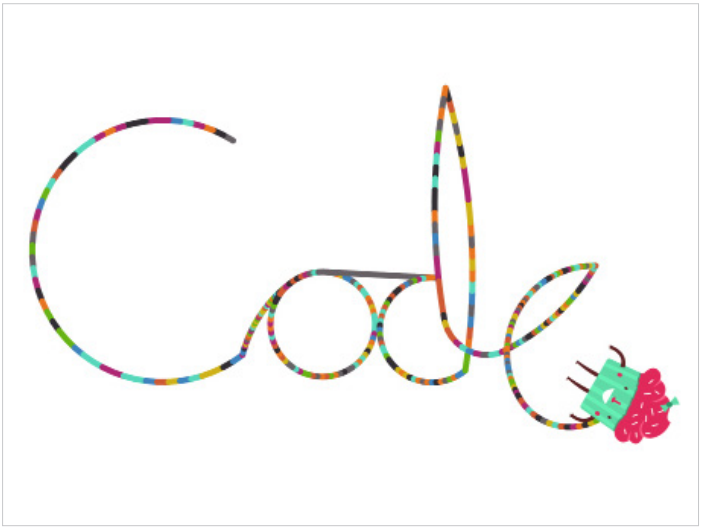



Random output for line colors and line width

15 min.

**Rando-Rainbow-Robo-Signature**

Have students edit their signature code to utilize random colors and pen thickness.



Rando-Rainbow-Robo-Signature

LESSON 5

Get Organized

## Teacher Brief

Now that you've been coding for a while, you may have noticed that sometimes code can be associated with an action. A very simple example might be the power button on your computer. When you press it, some code is run to turn a computer off or on. A more complex example is collision detection in Angry Birds; when a bird hits a block, that block should fall down. To associate a block of code with a specific action, we use a function called an "event handler."

Event handlers are a great way of organizing and generalizing code. For example, you might have a block of code that you want to run when the user taps anywhere on the iPad. Regardless of where they touch, the exact same code will run. If you had to duplicate that code for every possible spot the user could touch, you'd have a lot of repeated code for no reason.

The goal of this lesson is for to get students comfortable writing event handlers and to identify where event handlers are used in the apps and products they use everyday. Examples might include an event handler to bring up the dialing interface on a phone when the phone detects a finger touching a certain icon, or an event handler in a game to slash a piece of fruit when a finger has made a swiping motion.

## Curricular Goals

- Students are introduced to basic functions/ event handlers (e.g. the idea of repeated code). Students should feel comfortable with event handlers in Hopscotch.
- Age appropriate understanding that actions can be linked to a set of code (e.g. pressing a power button will run code to turn a computer off or on.)

## LESSON 5: Activities

SLIDES

5 min.

**Lesson 5 Slides: "Events"**

Use these slides to introduce event handlers and show how they can keep code clean and organized.

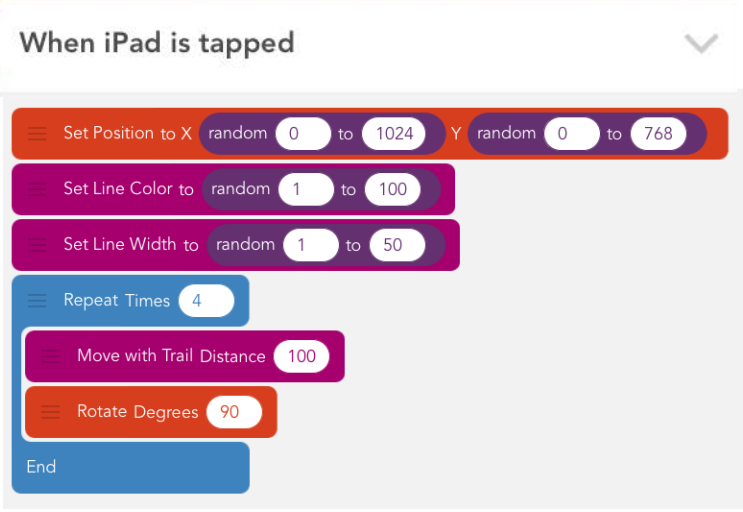
Show a visual example of beautiful functional code versus repeated code.

CODE DEMO

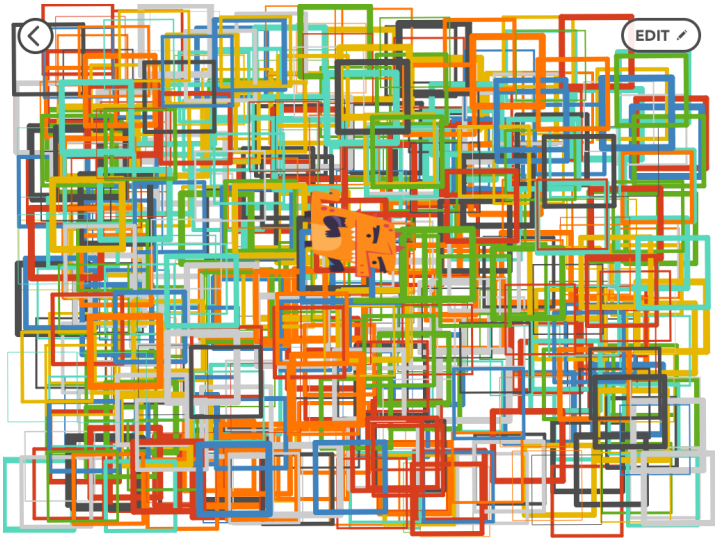
10 min.

## Random Squares Group Code

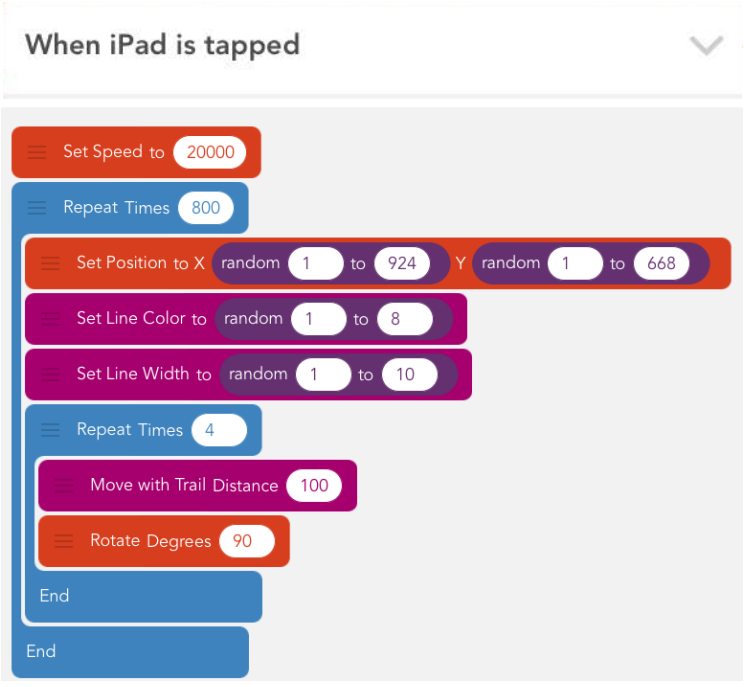
Walk through the Mondrian group code to draw a square in a random location when the stage is tapped. Once this is complete, update it so that the squares have a random color and thickness. The random bounds for x are 1 - 1024, and the random bounds for y are 1 - 768.




Mondrian Code




Mondrian Project



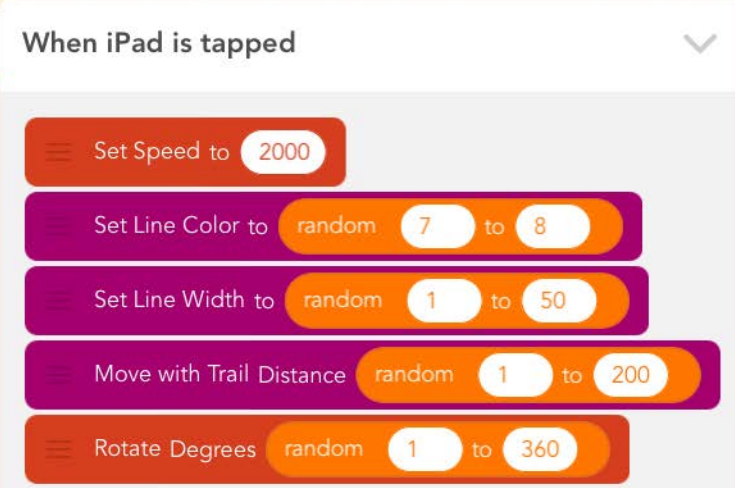

Mondrian Advanced Code

Should your students be advanced, you can also try this example code.

30 min.

**Jackson Pollock**

For the remainder of class, have students write a program to imitate Jackson Pollock style paintings. Each time the user taps the iPad, the object should move a random direction, painting a stroke of randomly colored paint of a random thickness. Attached is one possible implementation.




Jackson Pollock code


Jackson Pollock project

LESSON 6

# Put it All Together!

## Teacher Brief

You and your students have come a long way. You actually know nearly everything necessary to write any algorithm or program! Thus far, however, students have been working on smaller projects focused on one specific skill.

The goal of this assignment is for students to gain comfort working on a project where they must combine what they have learned, independently identify whether they need loops, event handlers, or just straightforward instructions, and develop a plan before they begin coding.

This problem is actually not harder than previous assignments, but requires more independence, planning, and abstract thinking skills as there is no introduction suggesting a way to solve the problem.

It is helpful to have students write down the process/algorithm they are trying to achieve in plain, colloquial English before they begin. Then have students read through what they wrote and try to identify the event handlers and loops. Once that is done, they should write their algorithm again by hand, but this time try to make it look more like Hopscotch. Once this is done, they are ready to code!

This process is extremely important as **computer science is not about coding, but rather about identifying why certain problems can be solved by computers and creating algorithms to solve them.** Only after these two steps are completed does actually coding become relevant.

Curricular Goals

- Students gain comfort writing a program from start to finish independently.
- Students should practice thinking about the problem and writing "pseudo-code" or a plan of how the code might work before they start coding.

## LESSON 6: Activities

5 min.

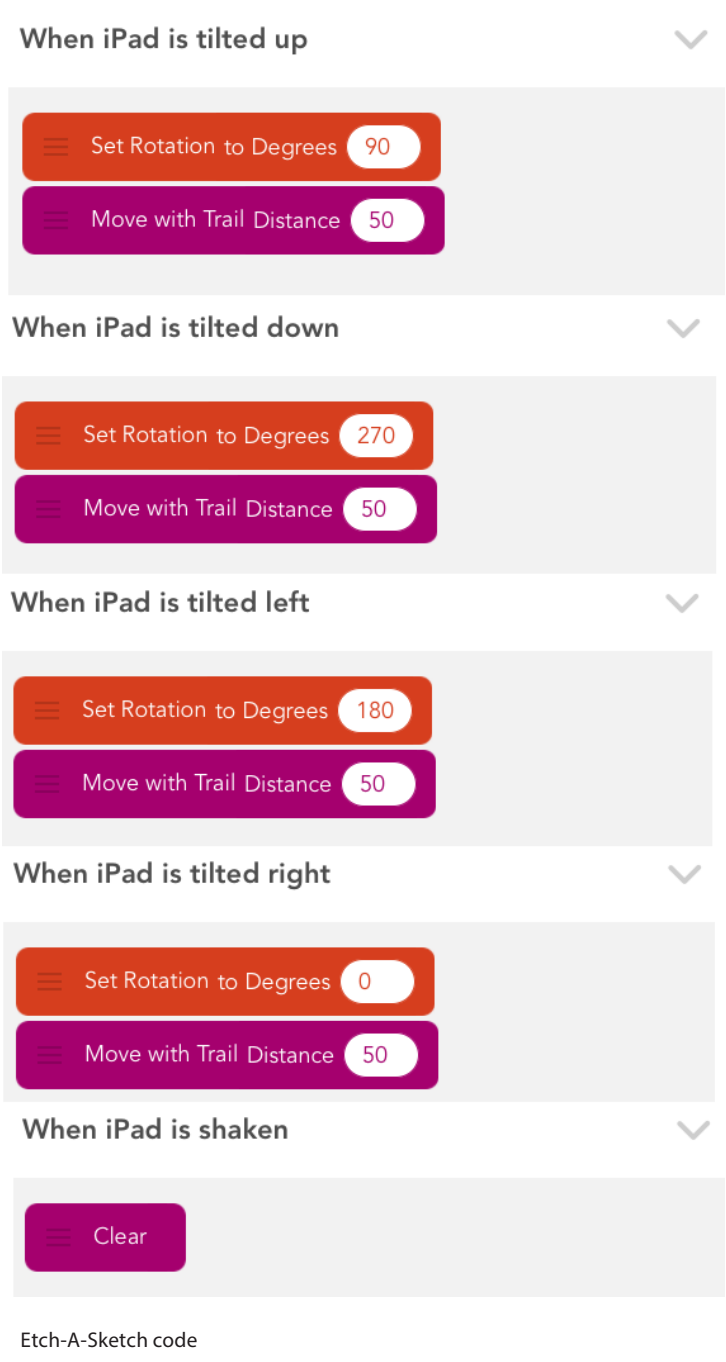
**Introduction To Etch-a-Sketch**

Ask your students if they know what an Etch-A-Sketch is and how to use it (they may be too young to have ever encountered it!) Explain how it works, and show your students a video of an Etch-A-Sketch masterpiece. Once complete, tell students that they will write their own Etch-A-Sketch for iPads! When the user tilts up, down, left, or right, the object should draw a line in that direction. When the user shakes the iPad, the screen should clear.

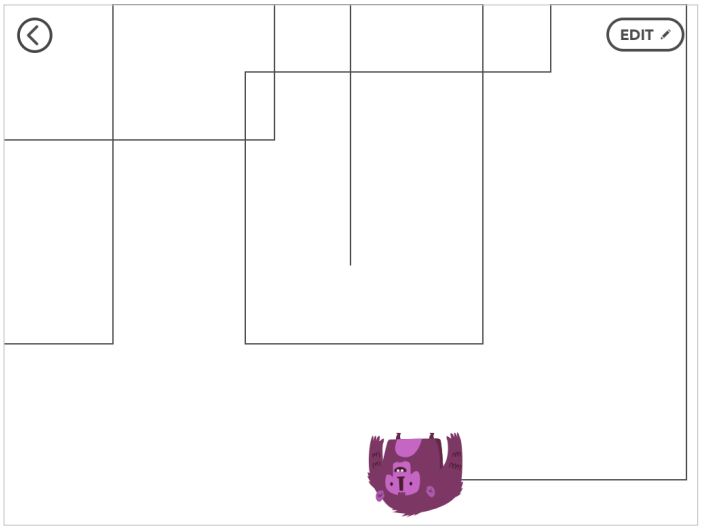
40 min.

**Etch-a-Sketch**

Students should use the remainder of class time to write their Etch-A-Sketch. Here is one possible solution and output:




Etch-A-Sketch code




Etch-A-Sketch project

LESSON 7

Final Project

## Teacher Brief

This assignment is largely to reinforce the curricular goals of the last lesson and to allow students to spend more time coding!

Additionally, students tend to enjoy assignments that they create themselves, without instructions, and therefore might love computer science a little more after this project.

The process of coming up with an idea that students can translate into code is an important one, as they must exercise the skill of identifying computable vs. non-computable problems (e.g. a video game vs. a mind reading program.) This assignment also gives students the chance to talk about their algorithms in front of the class.

## Curricular Goals

- Students gain comfort generating ideas and translating them into code independently.

## LESSON 7: Activities

40min.

**Students work on final projects**
Students work on a project of their choice!

Teachers, you could potentially add another class where students present their coding solutions and talk about them in a short, five minute presentation in front of the class.

SLIDES

5 min.

**Lesson 7 Slides: "Keep Coding"**

Use these slides to encourage your students to continue their exploration of programming.