HOPSCOTCH

HEY HOPSCOTCHERS,

We are so happy that you want to improve your Hopscotchin' skills!

This activity guide is made for everyone: parents, youth leaders, Hopscotchers, and you! In fact, we designed this tool to empower anyone to teach and learn about Hopscotch. You can use this guide with your child, a sports team, youth group, or classmates. If you're new to Hopscotch and want help practicing the basics, these activities will be your companion—you can even go through them on your own. For experienced Hopscotchers, this guide will expand your understanding of programming and inspire you to tackle new coding challenges.

These activities introduce and reinforce basic Hopscotch skills and, in the process, teach fundamental computer science principles. What's that, you ask?

WHAT IS COMPUTER SCIENCE AND WHY SHOULD I LEARN IT?

Computer science (also known as computer programming, computer coding or computer engineering) is a very misunderstood field. Many people think that using computer programs, like Microsoft Word or Google, is computer science. 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 you to make connections between events in history, computer science asks you to investigate the interactions between complex systems. When you write geometric proofs in math, we ask you 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 you to understand literal words as symbols and abstract entities in support of a larger theme, and in computer science we encourage you to take literal numbers and letters and endow them with complex meaning within a system.

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

--The Hopscotch Team

HOPSCOTCH

ACTIVITY GUIDE

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 improve your Hopscotchin' and learn computer science, we are providing you simple lessons. Each lesson is flexible: should it not fit your schedule or meet your needs, it is perfectly acceptable to cut it short or combine lessons into longer sessions as needed.

Tools:

- 7 flexible, 45 minute lessons. We recommend planning for 8 just in case things move slowly. Target age is 6th-7th grade, though Hopscotch is loved by kids as young as 6 and old as 65.
- Supporting slides
- The Hopscotch language and application (iPad only as of Sept 2014; download in the App Store)
- We include sample solutions to each coding challenge; you can take a peek if you're stuck or try writing your own first before reviewing ours.

No experience necessary, but we recommend you review the lesson overviews before tackling the activities.

LESSON 1

What is Computer Science? and Introduction To Hopscotch

Overview

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 you fully understands this concept, and begin to think of everyday processes (making a sandwich, getting to school) as a set of instructions, you 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 comfortable writing instructions in Hopscotch, and ensuring that you understand that the order of instructions matters.

Hospcotch uses some computer programming vocabulary that you may want to review 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.

Goals for this Lesson

- 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: Understand that computers can be programmed to solve problems. Examples include cell phones, computers, calculators.

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

  7th Grade: 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: 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

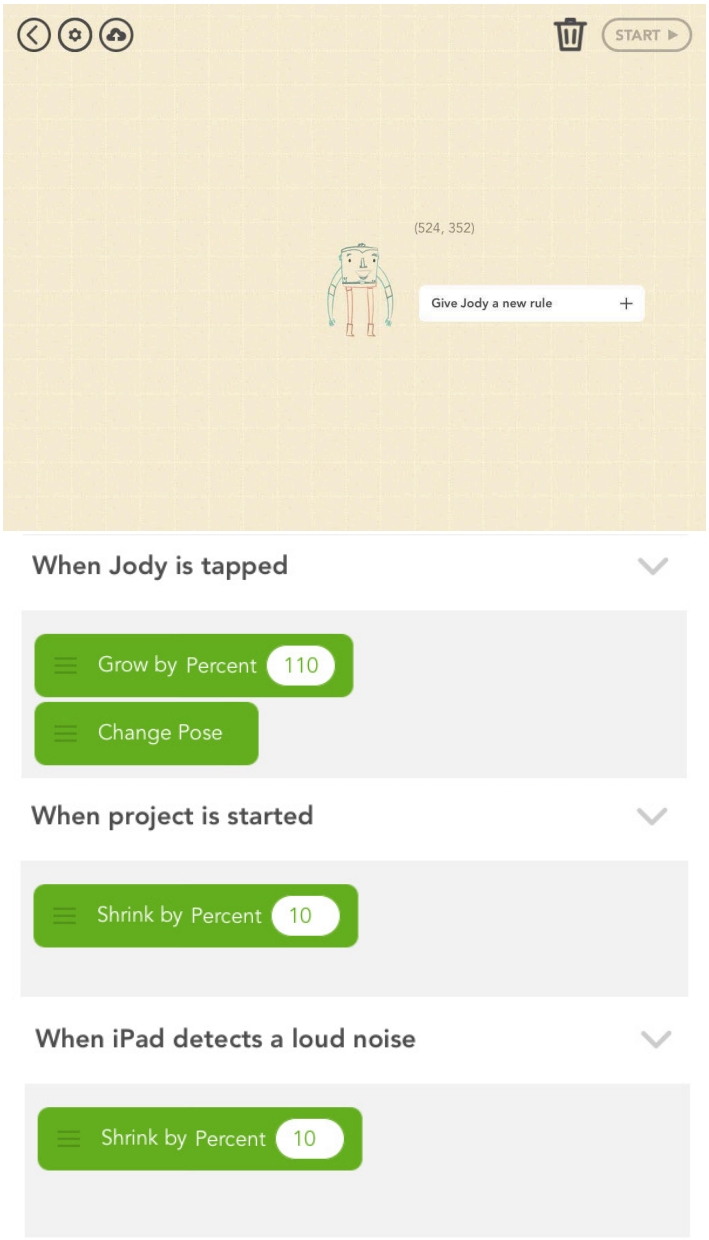
5 min.

Lesson 1 Slides: "What Is Computer Science?"
Review these slides to familiarize yourself with the concept of computer science.

CODE DEMO

5 min.

Monster Hugs
Review the below the Monster Hugs code. Then, try coding it in Hopscotch to see how it works. This is a simple but powerful example of how programming languages work.



Monster Hugs code

CODE DEMO

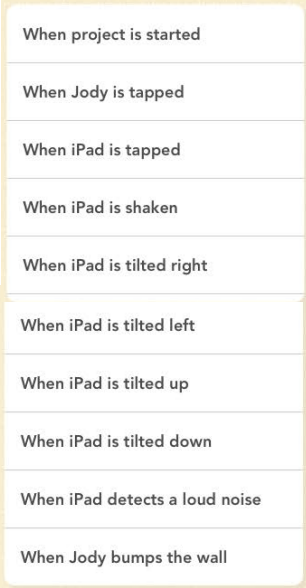
5 min.

Menu Walkthrough

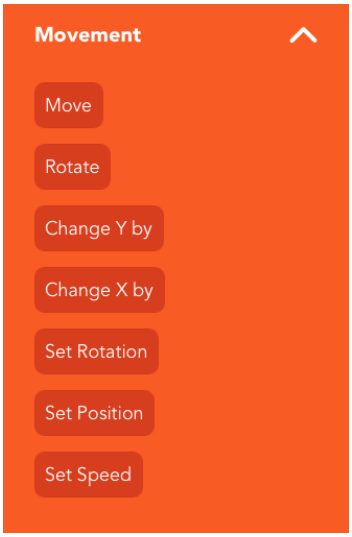
Create a new Hopscotch program and walk through each menu (Movement, Drawing, etc.) and the kinds of blocks they contain. Test some of the blocks in each menu by dragging them into the editor and pressing play to see what happens.

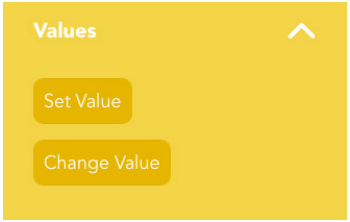
For example, use the move block to try out the Movement menu.

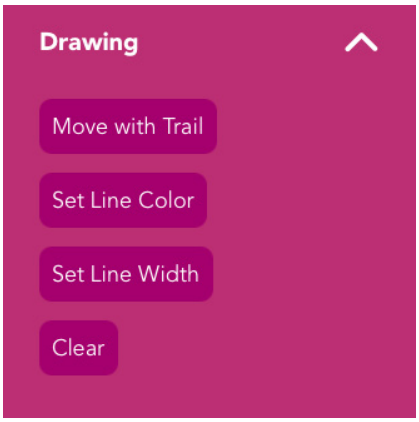
Event Handlers

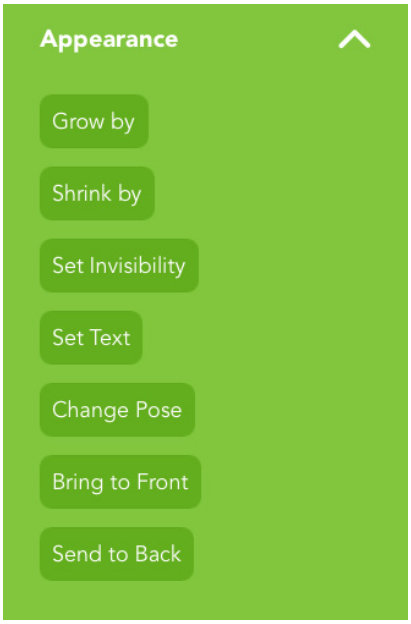


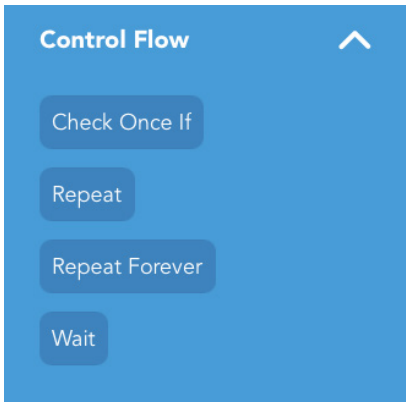
Menu Blocks











10 min.

Free Code

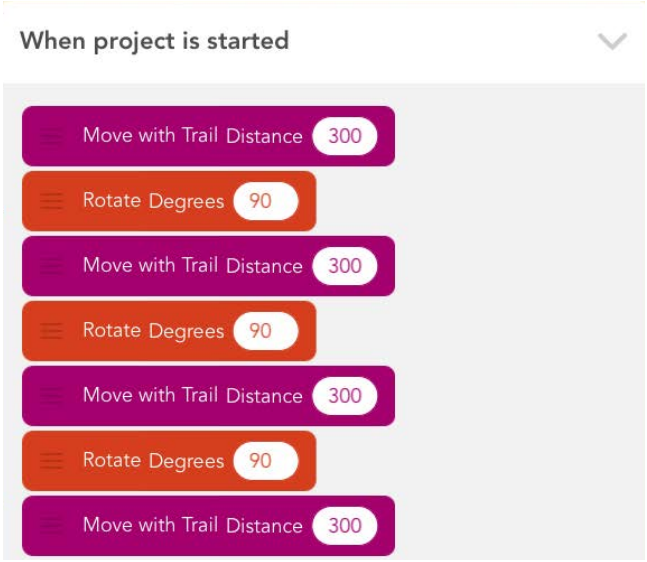
Take 10 minutes to explore and try out the app. Should you be inhibited, start by having your object (character) draw a line wherever it goes.

20 min.

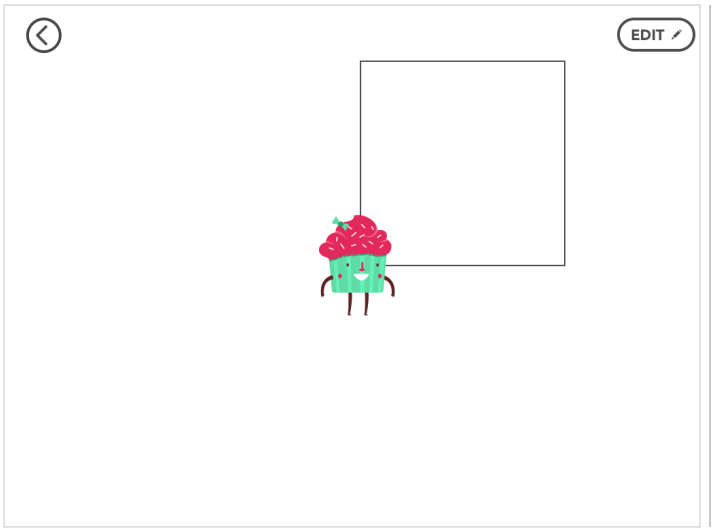
Square Lab

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

- Begin by pretending that you are the object and walk in a square motion.
- Record the steps.
- Investigate which blocks are most like the steps that you took.
- Begin coding using these blocks.



Square Lab code



Square Lab project

LESSON 2

Loops!

Overivew

As you 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" and "repeat forever" blocks.

Goals for this Lesson

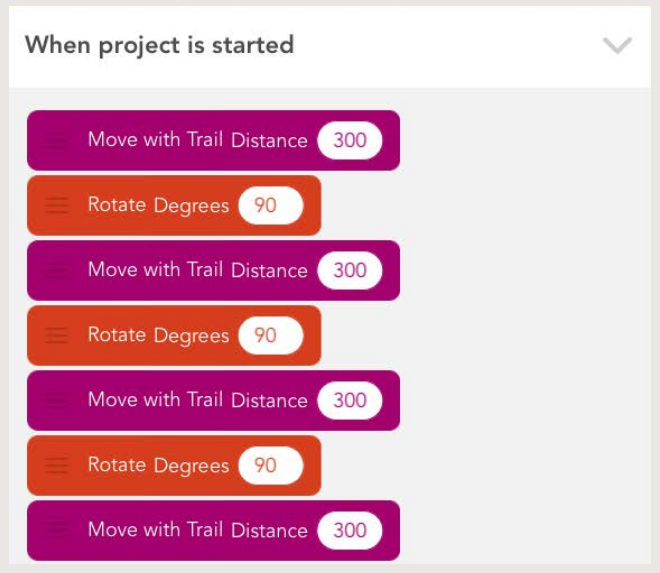
- Gain more comfort and practice with sequential thinking.
- Gain basic understanding of for-loop syntax.
- Basic introduction to abstraction: e.g. students can identify which lines of code represent lines vs. corners in a square.

CODE DEMO

5 min.

Discuss Square Solution

Walk through the solution code to previous lesson's square lab if further clarification is necessary. Review the commands that represent different parts of the shape (ie: move commands are representative of sides, and rotations are representative of corners). If you're doing these lessons by yourself, ask a parent or friend to review your code with you.



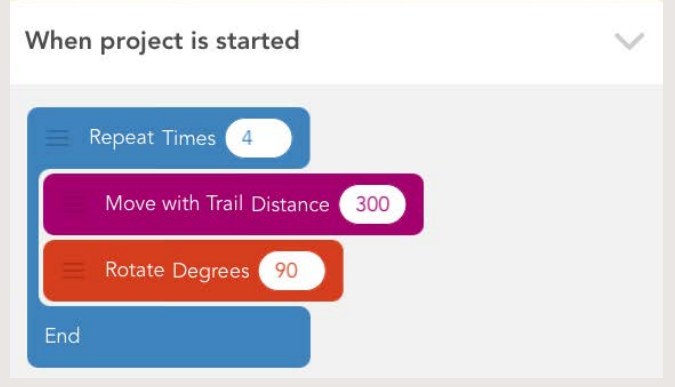
Square Lab code

SLIDES

10 min.

Lesson 2 Slides: "Loops!"

Familiarize yourself with loops using these slides.
Then walk through the square assignment using loops, as seen here, line by line.



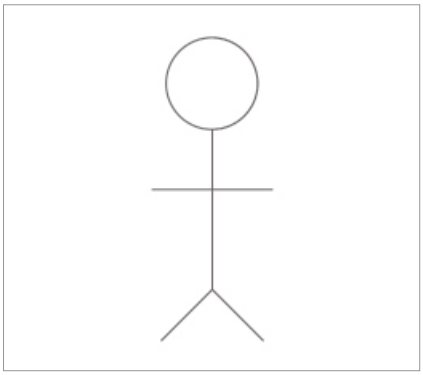
Square with loops

Use loops to draw a "Line Drawing" of your choice

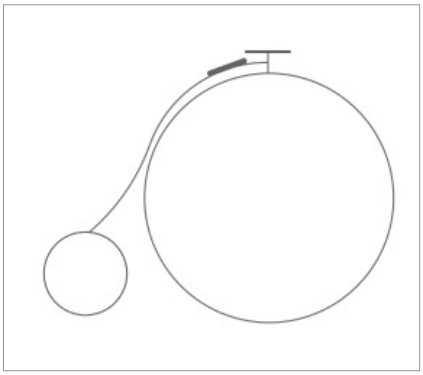
30 min.

Look up line drawings on the internet—drawings done using pen or pencil, and drawn in one continuous line , like this one. After doing so, try creating a line drawing in Hopscotch (like the below).

Star



Person



Bicyle

These are arranged in order of difficulty. 4-6th graders should be able to accomplish the first, 7-8th graders the second, and the third could be attempted by advanced students.

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

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

LESSON 3

Loops Continued!

Overview

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

Are there loops in the games that you 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 you perform in your everyday life (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 to understand the role of both of them, their interaction, and where each should be used.

Goals for this Lesson

- Achieve basic proficiency in for-loops and understand when and why they are used.
- Gain more comfort with very basic abstraction. Identify loops vs. sequential instructions in apps you use or games you 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

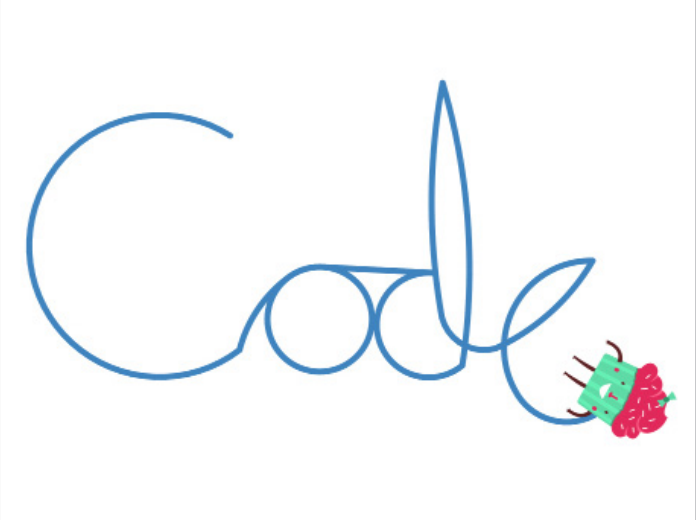
During this lesson you will write code to draw your signature in Hopscotch. Your should attempt to do so in cursive and not in block lettering. If this proves to be too difficult, or you think you need more time practicing with loops, you may use block lettering instead.

40 min.

Robo-Signature

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

If you are struggling with curves in loops, try the smaller challenge of simply making a circle. 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 have a perfectionist streak, try giving yourself a time limit for a first version and then perfect it later.



Robo-Signature

LESSON 4

Totally Random

Overview

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

Goals for this Lesson

- Introduction to computational thinking and algorithms. Understand that algorithms are recipes that computers use to solve problems.
- 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"
Familiarize yourself with randomness in nature, science, and computer science with these slides. (The video on the last slide is optional, as content may not be age appropriate.

15 min.

Maze Kinesthetic Activity
Let's move! Construct an algorithm to solve any maze:

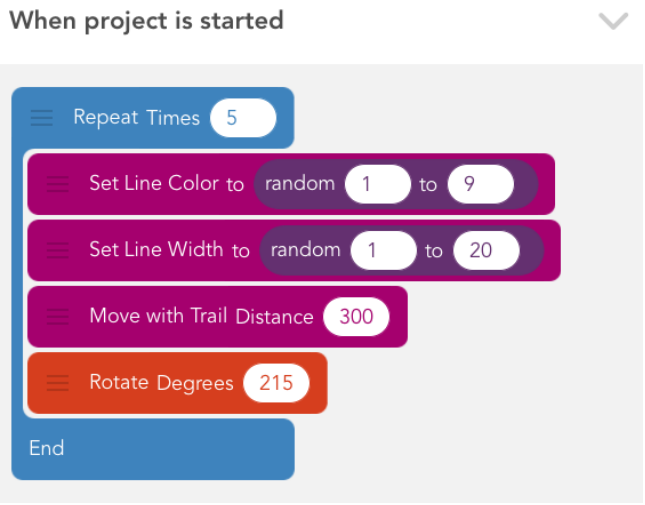
1. Begin by drawing a simple maze and writing instructions with a pen or pencil to get out of the maze.
2. Consider whether this same solution would work if you changed the maze.
3. Consider if randomness could be used to help someone get out of any maze while they were blindfolded. (e.g. move forward, turn a random direction -- given infinite time this will always work).
4. Working with another person, kinesthetically try the solution. Select a goal point in the room (e.g. the door) that you have to get to. Choose a random starting point across the room; distance from the goal does not matter.
5. 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 you have finished or time is up.
6. When complete, consider that distance from the door, or location, did not matter because you eventually reached the goal through random movements. If you or another player did not reach the goal, would you have eventually reached it?

CODE DEMO

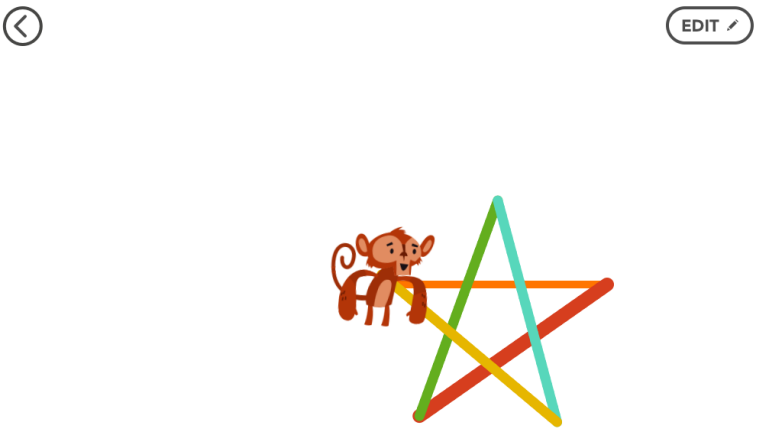
5 min.

Randomness in Hopscotch

Review some of the below examples of how randomness can be used in Hopscotch as provided here:



Random Code

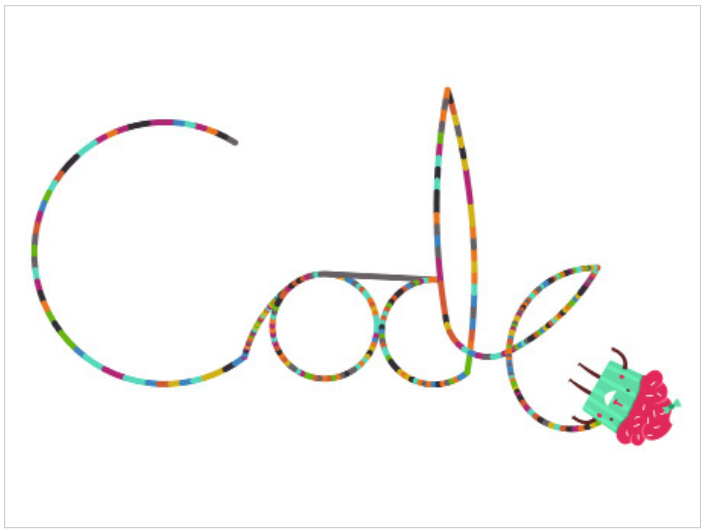


Random output for line colors and line width

15 min.

Rando-Rainbow-Robo-Signature

Edit your signature code to utilize random colors and pen thickness (hint: use blocks in your Appearance menu).



Rando-Rainbow-Robo-Signature

LESSON 5

Get Organized

Overview

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 to get comfortable writing event handlers and to identify where event handlers are used in the apps and products you 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.

Goals for this Lesson

- Introduction to basic functions/event handlers (e.g. the idea of repeated code). You 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

5 min.

Lesson 5 Slides: "Events"

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

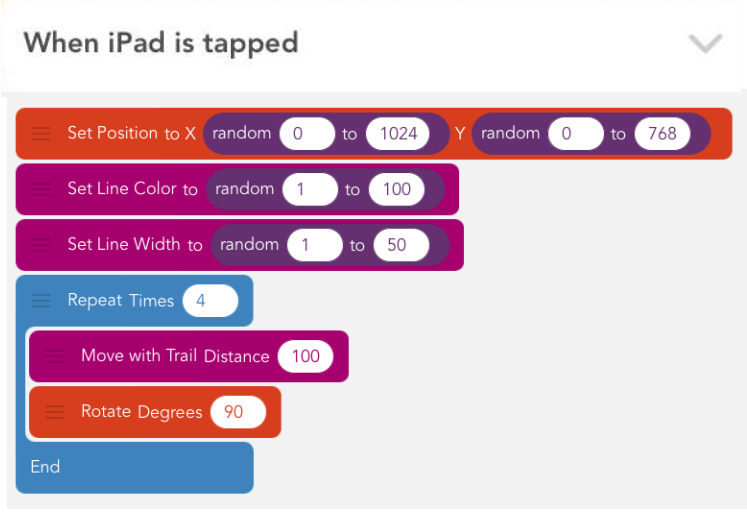
Review a visual example of beautiful functional code versus repeated code.

CODE DEMO

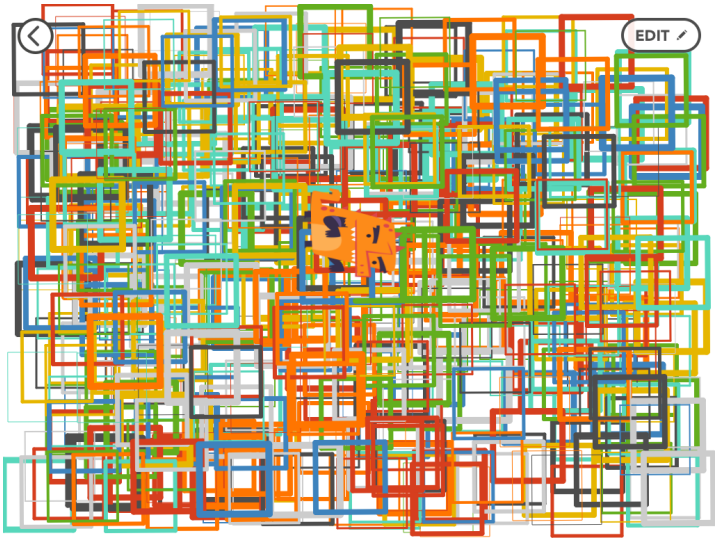
10 min.

Random Squares Group Code

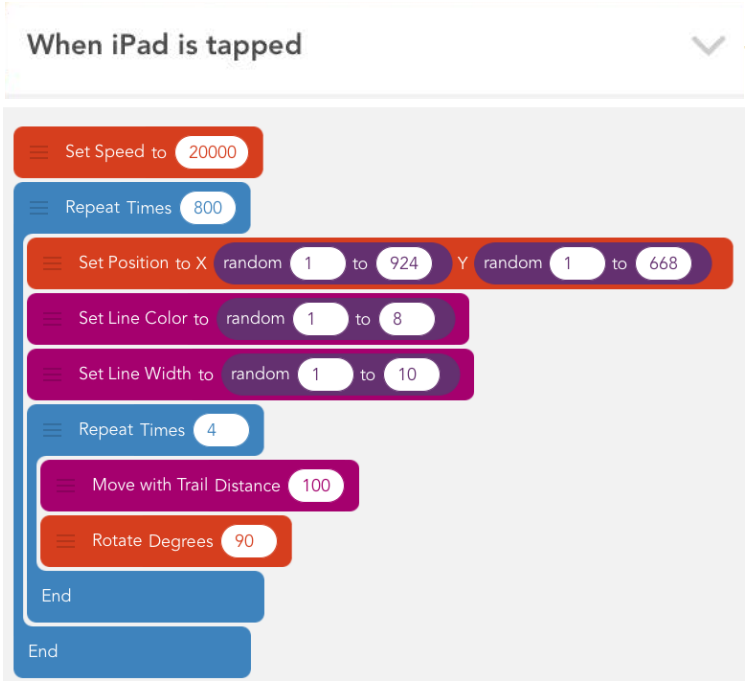
Review and try writing the below the Mondrian code to draw a square in a random location when the stage is tapped. Once this is complete, update it so that the squares have a random color and thickness. The random bounds for x are 1 - 1024, and the random bounds for y are 1 - 768.



Mondrian Code



Mondrian Project



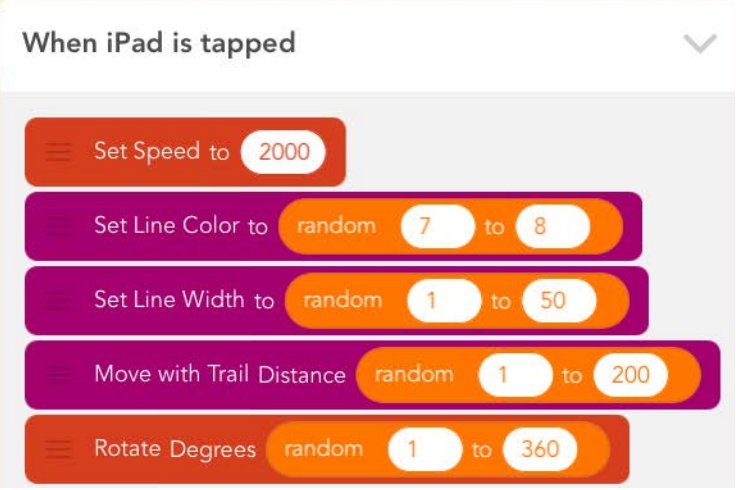
Mondrian Advanced Code

If you're more advanced, you can also try this example code.

30 min.

Jackson Pollock

For the remainder of the lesson, 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!

Overview

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

The goal of this assignment is to gain comfort working on a project where you must combine what you 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 write down the process/algorithm you are trying to achieve in plain, colloquial English before you begin. Then read through what you wrote and try to identify the event handlers and loops. Once that is done, you should write your algorithm again by hand, but this time try to make it look more like Hopscotch. Once this is done, you 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.

Goals for this Lesson

- Gain comfort writing a program from start to finish independently.
- Practice thinking about the problem and writing "pseudo-code" or a plan of how the code might work before you start coding.

LESSON 6: Activities

5 min.

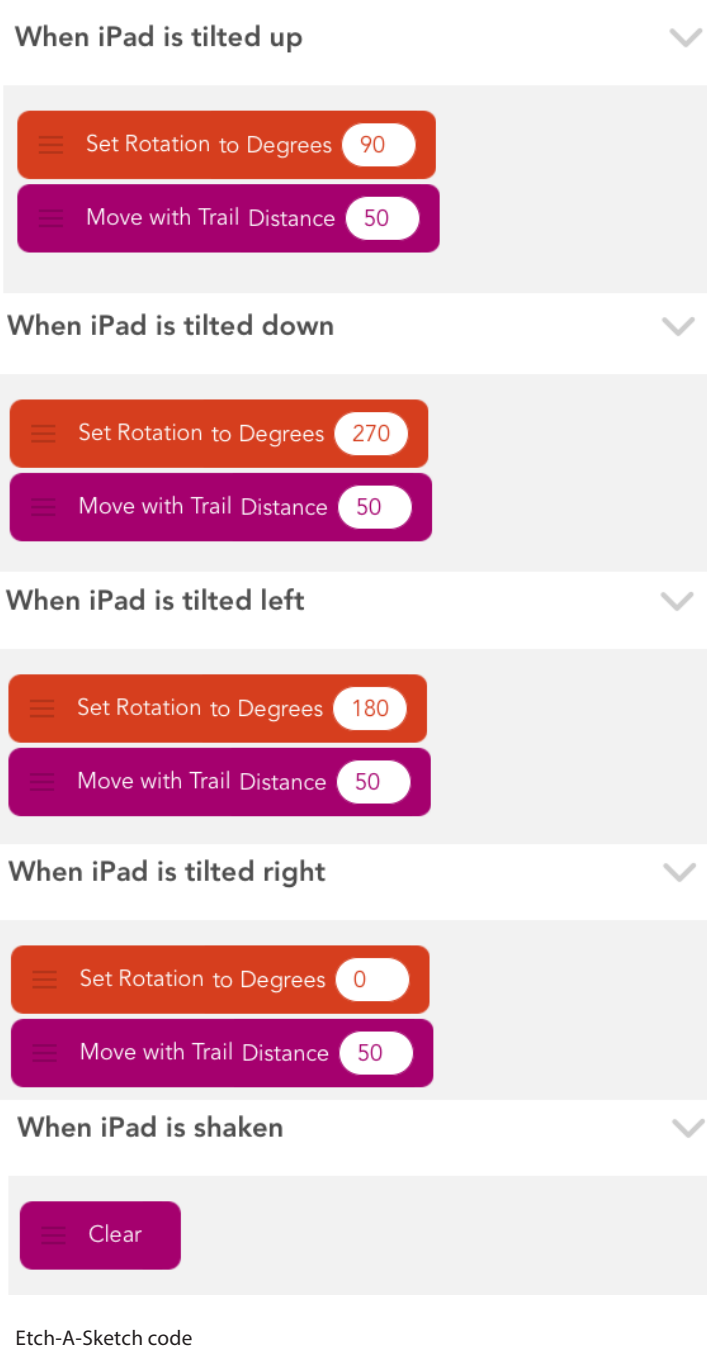
Introduction To Etch-a-Sketch

Do you know what an Etch-A-Sketch is and how to use it (old folks may remember using it as kids!)? Review how it works, and watch a video of an Etch-A-Sketch masterpiece. Once complete, you will write your 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.

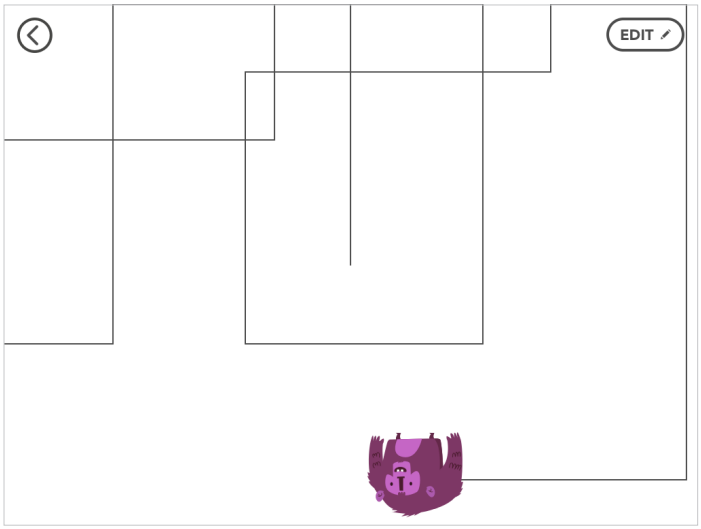
40 min.

Etch-a-Sketch

Use the remainder of the lesson to write your Etch-A-Sketch. Here is one possible solution and output:



Etch-A-Sketch code



Etch-A-Sketch project

LESSON 7

Final Project

Overview

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

Additionally, you may enjoy assignments that you create yourself, without instructions, and therefore might love computer science a little more after this project!

The process of coming up with an idea that you can translate into code is an important one, as you 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 you the chance to talk about your algorithms with others.

Goals for this Lesson

- Gain comfort generating ideas and translating them into code independently.

LESSON 7: Activities

40min.

Work on final projects Work on a project of your choice!

You could potentially add another lesson where you present your coding solution and talk about it in a short, five minute presentation in front of others (classmates, family, friends).

SLIDES

5 min.

Lesson 7 Slides: "Keep Coding"

Use these slides to encourage yourself to continue your exploration of programming.