Unit 3 - Intro to Programming

In Unit 3, students explore the fundamental topics of programming, algorithms, and abstraction as they learn to programmatically draw pictures in App Lab. An unplugged sequence at the beginning of the unit highlights the need for programming languages as well as the creativity involved in designing algorithms. Students then begin working in App Lab where they use simple commands to draw shapes and images using a virtual “turtle.” As they’re introduced to more complex commands and programming constructs, students learn to break down programming problems into manageable chunks. The unit ends with a collaborative project to design a digital scene.

Chapter 1: Intro to Programming

Big Questions

- Why do we need algorithms?
- How is designing an algorithm to solve a problem different from other kinds of problem solving?
- How do you design a solution for a problem so that is programmable?
- What does it mean to be a "creative" programmer?
- How do programmers collaborate?

Enduring Understandings

- 1.1 Creative development can be an essential process for creating computational artifacts.
- 1.2 Computing enables people to use creative development processes to create computational artifacts for creative expression or to solve a problem.
- 2.2 Multiple levels of abstraction are used to write programs or create other computational artifacts
- 4.1 Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- 5.1 Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- 5.2 People write programs to execute algorithms.
- 5.3 Programming is facilitated by appropriate abstractions.

Week 1

Lesson 1: The Need for Programming Languages

Students write instructions for building a small arrangement of LEGO® blocks and trade with a classmate to see if they can follow the commands to construct the same arrangement. The lesson derives the need for a well-defined programming language which leaves no room for interpretation.

Lesson 2: The Need for Algorithms

Students develop (and are eventually provided with) commands for a "Human Machine Language" designed to perform operations on playing cards. The lesson highlights the connection between programming and algorithms by showing that different techniques for solving the same problem can be expressed in the language.

Lesson 3: Creativity in Algorithms

Concept Invention | Unplugged | Algorithms

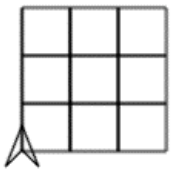
Students continue to work with the “Human Machine Language” - with an added SWAP command - to design an algorithm to move the minimum card to the front of the list. Students may design more algorithms for other problems and challenges provided.

Week 2

Lesson 4: Using Simple Commands

Turtle Programming | App Lab

Students use the App Lab programming environment for the first time and become acquainted with the turtle. The chief problem is to find the most “efficient” way to draw an image of a 3x3 grid using a limited set of only 4 commands.



Lesson 5: Creating Functions

Turtle Programming | App Lab

Students learn to define and call their own procedures (or “functions”) in order to create and give a name to a group of commands for easy and repeated use in their code. Named procedures are a form of abstraction that enable the programmer to reduce complexity by removing details and generalizing functionality.

Lesson 6: Functions and Top-Down Design

Turtle Programming | App Lab

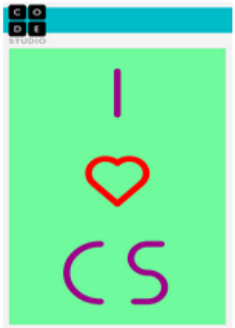
Students learn about top-down design strategies for solving more complex programming problems by breaking the problem down into small parts that can be named and represented as functions. The code in the resulting program should read more like a description of how to solve the problem than like raw code.

Week 3

Lesson 7: APIs and Using Functions with Parameters

App Lab

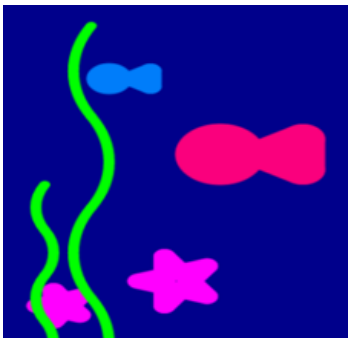
Students read and use App Lab's API documentation to learn about new turtle commands that they must use to complete a series of drawing puzzles.



Lesson 8: Creating Functions with Parameters

App Lab

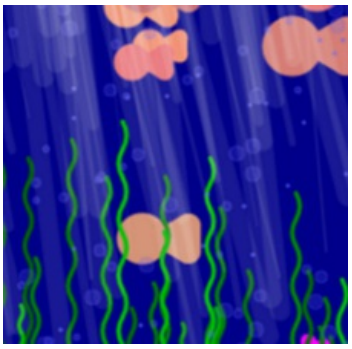
Students practice using and creating functions with parameters to generalize behavior that can vary. They make use of App Lab's randomNumber function to add variation to the scene.



Lesson 9: Looping and Random Numbers

Turtle Programming | App Lab

Students learn to use a simplified version of a for loop to add repetition to their code (i.e. repeat x times loop). Calling functions repeatedly with a loop combined with random numbers enables students to create more complex and varied drawings for digital scenes.



Week 4

Lesson 10: Practice PT - Design a Digital Scene

Turtle Programming | Project | App Lab

Students work in groups of 3 or 4 to design and write the code for a program that draws a digital scene of their choosing. Students break the scene down into small parts and divvy up the code writing amongst the team. Each individual's code is combined at the end to create the full scene. The project concludes with written reflection questions similar to those students will see on the AP® Performance Tasks.

Chapter Commentary

Key Concepts and Pedagogy

An Unplugged Introduction: Lessons 1 - 3 are designed to be fun, novel, and challenging for all students, without privileging students with prior programming experience. They introduce core ideas about programming and algorithms by letting students explore and discover them themselves.

Algorithm Design is Creative: In Unit 3, students will often want more commands than are provided. By intentionally imposing these limitations, these activities encourage students to creatively combine the limited commands available to solve problems. Often students will find that their approaches differ from those of their classmates while still solving the problem. This approach highlights the fact that algorithm design and programming are creative endeavors for which there are many possible “right answers.”

Programming as a Creative Outlet: Turtle programming allows students to quickly create fun digital drawings. This unit teaches many important programming concepts, but a primary goal is giving students tools to creatively express themselves with programming.

Top Down Design: Top Down Design is the process of breaking down a large problem into layers of successively smaller problems that are easier to solve. The solutions to these smaller problems can then be combined to solve the original large problem. In this unit students learn how to use Top Down Design to break down larger programming tasks, as well as the programming concepts that enable this approach.

This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 1: The Need for Programming Languages

Overview

At the beginning of a new unit we jump right into an activity - building a small arrangement of LEGO® blocks and then creating text instructions a classmate could follow to construct the same arrangement. Groups will then trade instructions to see if they were clear enough to allow reconstruction of the original arrangement. The wrap-up discussion is used to highlight the inherent ambiguities of human language and call out the need for the creation of a programming language which leaves no room for interpretation.

Purpose

This is the first in a 3-lesson sequence in which we attempt to make connections between programming languages, the creative act of programming, and algorithms.

The main activity is a derivative of a classic strategy for beginning to learn about programming - writing precise instructions for another person to do just about **anything** - making a peanut butter and jelly sandwich, a simple line drawing, an arrangement of objects - is challenging. Human language is fraught with ambiguities and assumptions that machines (computers) simply cannot handle.

Thus, the first step we take is to establish the need for programming languages. Writing down a precise set of instructions to build a small lego structure, as we do in this lesson, is nearly impossible…unless you can agree on certain language constructs ahead of time, and how they will be interpreted. **The main purpose of the lesson is to expose why programming languages are necessary, and how they come into being**.

When you formalize language or commands that describe actions you are making a kind of code. This is also necessary for computers, which are simply machines that can perform a number of different tasks. In order to write instructions for them to do something you must agree on the “code” and each action must have a precise, unambiguous meaning. **This is a programming language.** Novices might think that a programming language looks like an archaic, impenetrable mass of abstract word groupings, but all programming languages are derived from the **human** need to concisely give instructions to a machine.

Agenda

Getting Started (2 mins)

View on Code Studio

Objectives

Students will be able to:

- Assess the clarity of a set of instructions expressed in human language.
- Create a set of instructions in human language for building a simple LEGO block arrangement.
- Identify connections between the ability to program and the ability to solve problems.
- Describe the ambiguities inherent in human language and the ways programming languages seek to remove those ambiguities.

Preparation

- [ ] Blocks for the class (~5 per team/student)
- [ ] Paper for recording instructions

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **You Should Learn to Program: Christian Genco at TEDxSMU** - Video
- **LEGO Instructions** - Activity Guide **Make a Copy**
- **Building Blocks of Drawing** - Activity Guide **Make a Copy**

- Welcome to Unit 3!
- Activity (25 mins)
  - LEGO Instructions Activity
- Wrap Up (15 mins)
  - Discussion: Why is writing instructions hard?
  - The Need for Programming Languages
- Assessment
- Extended Learning

Teaching Guide

Getting Started (2 mins)

Welcome to Unit 3!

Remarks

> - Welcome to Unit 3 - Introduction to Programming
> - This unit is an introduction to the **principles** of programming.
> - As you'll see one of the most important things you can do in programming starts well before you write any code. It's about how you think.
> - We're going to launch into an activity right now that reveals an important principle of programming.
> - Try your best, and we'll discuss at the end.

Warm

This is the beginning of a new unit. Set the tone that as a class we're embarking on a new adventure - programming - and get right into the activity. Ask students to dive into working on the problem, and assure them we'll explain things at the end.

Teaching Tip

Students may be eager to program, but recognize that the course is about "principles" of programming, and we want to establish some of the thinking habits that people who are "good" at programming seem to have "naturally." There is nothing natural about it - these ways of thinking, these insights, are learned and practiced. And this thinking capabilities start here.

Activity (25 mins)

Distribute: Activity Guide - LEGO Instructions - Activity Guide (Non-LEGO Alternative: Building Blocks of Drawing - Activity Guide)

- Students can record their instructions on a plain sheet of paper, poster paper, piece of construction paper, etc.
- Students are encourged to work in groups of 2 or 3 (but can work alone)
- Each group should be given 5-6 LEGO® blocks.

LEGO Instructions Activity

Below are the steps students are asked to complete in the activity guide.

Teaching Tip

To make things interesting, strike a balance between all blocks being different colors and all blocks being the same. This forces students to be precise in identifying which block should be used in each step.

> **Create a simple LEGO arrangement (and record it)**
>
> Groups should create an arrangement of their blocks in accordance with the following rules:
>
> - All pieces must be connected in a single arrangement.
> - Color matters: the final arrangement must be absolutely identical to the original.
> - Groups should record their arrangement by taking a picture or creating a simple drawing.
>
> **Write instructions for building your arrangement**
>
> On a separate sheet of paper, each group should write out a set of instructions, that another group could use to create the same arrangement. A couple of guidelines are below:
>
> - You may only use words when creating these instructions. In particular, you may not include your image or draw images of your own.
> - Try to make your instructions as clear as possible. Your classmates are about to use them!
>
> **Trade instructions and attempt to follow them**
>
> Groups should disassemble their arrangements, place their instructions next to the pieces, and hide their image or drawing of the arrangement somewhere it cannot be seen. Have groups move around the room to other groups' instructions and try to follow them to build the desired arrangements. Have the original group check whether the solutions

are correct or let groups check their solutions themselves by looking at the recorded image of the arrangement afterwards. Allow groups an opportunity to try a few of their classmates' instructions before reconvening.

Wrap Up (15 mins)

Discussion: Why is writing instructions hard?

Bring the class back together to reflect upon the following prompts or similar ones of your own creation. You may wish for students to record their answers in writing.

Prompts:

- **"Were you always able to create the intended arrangement? Were your instructions as clear as you thought?"**
- **"Why do you think we are running into these miscommunications? Is it really the fault of your classmates or is something else going on?"**

Discussion

- Highlight the fact that ambiguity in human language led to issues or at least difficulty in creating the arrangements.
- Motivate the need to create a well-defined set of commands that all parties can agree upon for expressing the steps of a task, or in other words, a programming language.

In the discussion of these prompts, ask students to mention instances where language was ambiguous or detail was not as great as students thought. Establish the fact that even when we are very careful, the very nature of human language means there is typically some room for interpretation. In day-to-day life that's fine, but when expressing algorithms we need to do better.

Brainstorm: Transition the conversation to a more proactive response to the day's activity. Ask students to brainstorm ways to overcome the ambiguity of human language.

Prompt:

- If we were going to change human language to improve our odds of succeeding at this activity, what types of changes would we need to make?

Let students discuss at tables before again sharing with the class. Key ideas to bring out are:

- So long as there are multiple ways to interpret language, we cannot have perfect precision.
- If we rigorously define the meaning of each command we use, then we can avoid misinterpretation and confidently express algorithms.
- This is different from the way we normally think and talk, and it might even take a while to get comfortable with communicating in this way.

The Need for Programming Languages

Remarks

> Today we saw how human language may not always be precise enough to express algorithms, even for something as simple as building a small LEGO arrangement. The improvements you have suggested actually create a new kind of language for expressing algorithms, which we as computer scientists call a programming language. In the coming unit we are going to learn a lot more about how we can use programming languages to express our ideas as algorithms, build new things, and solve problems.

Teaching Tips

Students may say things like "We could be really careful about our language" or "We could include more detail." These are entirely true, but emphasize that if there's room for interpretation then we can't be certain about our results. Move towards the solution of creating a new language (a programming language!) in which we have strictly and precisely defined exactly what every word means. This is different from our normal human language, but necessary if we want perfect clarity in expressing instructions.

Assessment

Prompts:

- Consider the algorithm you designed for today's activity. Identify two instances where there may be multiple ways to interpret your instructions and suggest improvements that could be made to improve their clarity.
- Describe the features of a programming language that make it different from the language you are used to using in everyday life. Explain why a programming language must be created in this way.

Extended Learning

Ask students to rigorously define a small set of commands that could be used to build any arrangement of LEGO blocks. For example: "rotate", "attach", "detach", "move", etc. Feel free to let students select their own and then test out their new and more precise set of commands by repeating today's activity using them.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **4.1** - Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **5.2** - People write programs to execute algorithms.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 2: The Need for Algorithms

Overview

This is the 2nd day of a 3-lesson sequence in which we attempt to show the "art" of programming and introduce the connection between programming and algorithms. In the previous lesson we established the need for a common language with which express algorithms to avoid ambiguity in how instructions would be interpreted. In this lesson we continue to establish the connection between programming and algorithms, with more emphasis on the "art" of algorithms.

First students are presented with a new task for the “human machine” - to write a set of instructions to identify the smallest (lowest value) card in row of cards on the table. Once again we try to establish a set of fundamental commands for doing this, and develop a more formal set of “low-level” commands for manipulating playing cards. Students are presented with a "Human Machine Language" that includes 5-commands and then must figure out how to use these primitive commands to “program” the same algorithm.

At the conclusion several points about programming can be made, namely: 1. Different algorithms can be developed to solve the same problem 2. Different programs can be written to implement the same algorithm.

Purpose

The main point here is to connect the acts of writing "code" and designing algorithms, and to take some steps towards programming with code. To do this we imagine trying to write instructions for a "Human Machine" to complete a tightly defined task with playing cards.

We want to introduce the term algorithm and what designing an algorithm means in computer science (i.e. programming). We then want to take a few steps to build up to writing an algorithm with "code". Here are the basic steps of the lesson and their underlying purpose.

Step 1: Discover common instructions

When students write instructions to find the minimum card in a row of cards, we'll discover that even though the words used might be different between groups, there is probably a lot of commonality to underlying actions we need (shifting hands, picking up cards, comparing cards, jumping to a certain line in the instructions, etc.)

Step 2: Agree on a minimal instruction set

Recognizing these commonalities we can give names to a few commands and come to agreement about how they should be

View on Code Studio

Objectives

Students will be able to:

- Trace programs written in the "Human Machine Language"
- Develop an algorithm to find the smallest playing card in a row of cards
- Express an algorithm in the "Human Machine Language"
- Identify the properties of sequencing, selection and iteration the "Human Machine Language"
- Evaluate the correctness of algorithms expressed in the "Human Machine Language"

Preparation

- ☐ **KEY Human Machine Language**
- ☐ **KEY Minimum Card Algorithm**

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Human Machine Language** - Activity Guide **Make a Copy**
- **Minimum Card Algorithm** - Activity Guide **Make a Copy**

Vocabulary

- **Algorithm** - A precise sequence of instructions for processes that can be executed by a computer
- **High Level Programming Language** - A programming language with many commands and features designed to make common tasks easier to program. Any high level functionality is encapsulated as combinations of low level commands.
- **Low Level Programming Language** - A

interpreted. This is the foundation for a "code". We then provide a 5-instruction "human machine language" that is sufficient for implementing an algorithm to find the minimum card.

Step 3: Use the provided Human Machine Language "code" to implement an algorithm

The art of programming is to solve computational problems using a provided language that controls the machine. In the activity we provide a simple, low-level, language for acting on playing cards. The point is to show that even when you know what the commands are, you still need to be creative to use them to solve a problem.

Why the Human Machine Language? This language bears a strong resemblance to so-called **"low level" programming languages** - a sparse, primitive set of commands to directly control the physical/electronic operations of a computing machine. Other programming languages are built on top of the low level languages to provide more abstraction and functionality that combines low level operations into commonly used procedures. The most commonly known low level language is called **Assembly Language** For a good example **see the wrap up of the next lesson** where we introduce students to low level languages.

Agenda

Getting Started (5 mins)

- **Recall the lessons learned about language**
- **Define: Algorithm**

Activity 1 (30 mins)

- **Find Min Card Algorithm**
- **Discuss - Define a language**

Activity 2 (30 mins)

- **The "Human Machine" Language**
- **Challenge: Find Min with the Human Machine Language**
- **Share solutions to Find Min with the Human Machine Language**

Wrap Up (10 mins)

- **Discuss - The "Art" of Programming**

programming language that captures only the most primitive operations available to a machine. Anything that a computer can do can be represented with combinations of low level commands.

Teaching Guide

Getting Started (5 mins)

Recall the lessons learned about language

Yesterday's activity focused on the inherent difficulties of trying to express precise processes with written language. We arrived at a few conclusions...

- We need to agree on a set of commands and exactly what terms mean
- The fewer commands we have, the easier it is to agree
- We want to know what are the "primitive" operations - the most basic set of operations that will allow us to do most of the tasks that the situation requires.

Define: Algorithm

Remarks

> - Language is important, but there is another part to programming. Once you have a well defined language you need to apply it to problems.
> - The art (and science) of using a well-defined language of primitive operations to solve problems is the art and science of **algorithms**.
> - The CS Principles definition of algorithm is: **Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.**
> - One way to think of the study of algorithms is that it is the**study of processes** -- how can you use a small set of instructions to clearly and correctly define process that will solve some problem?
> - Yesterday, with the LEGO blocks, you also attempted to design an algorithm. Any time you are trying to write a precise set of instructions for a process to solve a problem you are designing an algorithm.
> - Today we're going to get into algorithms a little more deeply.

Activity 1 (30 mins)

Find Min Card Algorithm

Remarks

> Perhaps it goes without saying that in a Computer Science class we are concerned with not just any processes, but computational processes - ones that can be executed by a computer - which have specific sets of constraints.
>
> We often get started thinking about algorithms and computational processes by trying to rigorously act them out ourselves as a sort of **"Human Machine"**. When acting as a machine, we can keep the limitations of a computer in mind.
>
> In this activity you're going to pretend that you are a**"Human Machine"** that operates on playing cards on the table.

Distribute Minimum Card Algorithm - Activity Guide

Put students into pairs

> After distributing the guide and playing cards:
>
> - Get clear on the task, rules, instructions
> - With a partner act out an algorithm
> - Write down the steps
> - Give students time to work

As students are working you might ask probing questions like:

- **"How do you know when to stop?"**
- **"Do your instructions state where and how to start?"**
- **"Is it clear where to put cards back down after you've picked them up?"**

Discuss - Define a language

- Invite a pair to share their solution by reading it out loud or displaying.
- Ask if any other groups did something similar.

Prompt:

- **"As we look at these algorithms you came up with, we can see they are not all the same. However, there are common things that you are all making the human machine do and commonalities in some of your instructions. Can we define a language of common Human Machine commands for moving cards around? What are the commands or actions most of these instructions have in common?"**
- Ask students to suggest commands
- Write them down or group them based on type.
- Hopefully, you can derive a set similar to what we'll use in the next activity:
  - **SHIFT** (hand) - some form of shifting hands one position down the row left or right
  - **MOVE** (hand) - some form of moving a hand directly to a particular card based on its position in the list or to the position of one of the other hands.
  - **COMPARE** - some way to compare cards and do something based on the result like: "if card in right hand is less than card in left hand then..."
  - **GO TO LINE** - some way to jump to an earlier or later line in the program
  - **PICK CARD UP/PUT CARD DOWN** - some way to do this that also makes clear where to put a card back down. Typically something like: "Put right hand card down into the right-most open space in the row of cards" **NOTE:** we don't need this command for the next activity so just acknowledging the need is fine.

Transition to next activity...

Teaching Tip

Circulate the room asking pairs to demonstrate what they are coming up with.

Use the suggested solution in the answer key, as well as the notes on the discussion that follows this activity, to help guide your questioning.

You're looking for students to:

- Develop deeper understanding of the problem - trying to write an algorithm should evoke lots of questions about details like: "where do hands start?" "can we number things?"
- Develop clear, precise processes - ask questions to address gaps in clarity. Students might have to define or make up their own terms for things.

Discussion

The goal here is to define and agree upon a language for moving cards around.

You'll probably need to steer the conversation a little bit so that you land on the commands we need for the upcoming activity. You can probably coerce the conversation based on your observations circulating the room.

A secondary, longer range, goal here is to see where programming languages come from - it's often from a process just like this. What can end up looking cryptic is often actually simple, or at least stems from trying to keep things as simple as possible.

Activity 2 (30 mins)

The "Human Machine" Language

We have just identified a set of primitive commands that we can use to operate on a set of cards.

To be very concrete let's formalize this into a language...

Distribute Activity Guide - **Human Machine Language - Activity Guide**

Introduce Human Machine Language

- Have students read the first page.
- Clarify the instructions and setup.
- Put students into partners (same partner they developed the Find Min algorithm with)

Students Execute the Example Programs

- Students should **try to figure out the example programs with a partner** , making use of the code reference guide.
- One partner reads, the other acts as the human machine.
- They should jot notes about what the program does.

Review the example programs and answer questions (See: **KEY - Human Machine Language - Answer Key**)

- Verify that students get the gist of how the language works.
- For each example you can ask students to describe what it did.
- Review the last example which had a problem.

1. Make sure everyone understands the problem
2. Have students suggest a solution (or show a solution along with the following transitional comments)

Transition to the challenge

> This problem we identified with the last example speaks to the art and science of algorithm design and what can make it so interesting.
>
> The question is: can we fix the problem without adding more commands to the language? Yes. (see examples of fixes)
>
> If we can fix a problem without extending the language, that's a good thing. We can focus our attention on designing algorithms given these constraints.
>
> Let's try to write FindMin using the Human Machine Language...

Challenge: Find Min with the Human Machine Language

First identify what's different about the problem setup for the Human Machine Language:

- All cards are face up
- Card positions have numbers
- Don't need to pick up cards or put them down
- There is actually no way to move cards at all - only hands
- The ending state is well defined - left hand touching the min card.

Now use the Human Machine Language to write the algorithm for finding the min card.

- NOTE: Students **can** just write the code, or you can use the cutout strips of the commands and write values into the boxes.
- Give students time to work things out with their partners.
- It may take some time to get oriented and understand the task.

Teaching Tip

Reviewing Differences

- This might be **very** challenging at first since the problem setup is slightly different, which is why it's worth reviewing these differences before students set out on the challenge.
- The problem now has different initial assumptions.
- It's likely that student's hand-written instructions will differ because:

1. They weren't restricted to one command per line as they must be for the Human Machine Language
2. It involved actions like picking up cards, changing hands, or moving cards to other locations, which is also not possible with the Human Machine Language.

Don't need to finish today

- It's okay if you can't quite finish before the period is over. This **activity continues in the next lesson** anyway. You can allow this to overflow, and make the wrap up points later.

Share solutions to Find Min with the Human Machine Language

- Put pairs together to compare solutions.
- Each group should test out the other group's code by acting as the human machine.

- During the comparision note any differences in people's approach.
- There are several ways to go about it but the canonical solution is given in the answer key.

Share out goal

- This is a first experience in evaluating some code.
- It's important to actually try out a solution to identify any potential errors.
- It's also important to see that people's programs are **different**
- See the wrap-up notes for other things to identify as students share

Wrap Up (10 mins)

Discuss - The "Art" of Programming

Connect algorithms to programming.

- Yesterday we discussed the need for a programming language
- Today we came up with our own programming language and used it to implement an algorithm.
- The CSP definition of algorithm is: “a precise sequence of instructions for processes that can be executed by a computer and are implemented using programming languages."

Notice two things about algorithms and programming...

1. **Different algorithms can be developed to solve the same problem**

   Even though you were all trying to solve the same problem (find min) as a class we came up with different methods for doing it. We would say we came up with different algorithms.

Content Corner

Point 1 is verbatim from the CSP Framework -- Essential Knowledge statement 4.1.1H

Point 2 is not in the framework, but is equally true.

2. **Different code can be written to implement the same algorithm**

   This is sometimes surprising to newcomers. When writing "code" (even with with the human machine language) two people trying to write code to implement the **same algorithm** may very easily code it differently.

Remarks

> **These two facts** - Different algorithms can be developed to solve the same problem and different code can be written to implement the same algorithm - **embody art of programming** and what makes programming so fun, engaging and creative.
>
> In programming, just like art, we strive to make beautiful things:
>
> 1. A beautiful algorithm is an elegant and clever idea for how to solve a problem.
> 2. A beautiful program is an elegant use of whatever language structures are provided to make the algorithm actually work on a computer.

Foreshadow: tomorrow we'll try some other algorithms in the human machine language.

Standards Alignment

Computer Science Principles

- **4.1** - Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **4.2** - Algorithms can solve many but not all computational problems.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 3: Creativity in Algorithms

Concept Invention | Unplugged | Algorithms

Overview

This is the third of three lessons that make the connection between programming and algorithms. In this lesson students continue to work with the "Human Machine Language" to get creative designing more algorithms for playing cards. One command is added to the language from the previous lesson (SWAP) that allows positions of cards to change. With the addition of swap the challenge is to design an algorithm that will move the minimum card to the front of the list while keeping the relative order of all the other cards the same. If that is achieved some other Human Machine Language challenges are available.

Purpose

The purpose of this lesson is to see what "creativity in algorithms" means. Creativity has to do with both the process you invent (an algorithm) to solve a new problem in a given context **and** how you implement that algorithm in a given language. Creativity often means combining or using algorithms you know as part of a solution to a new problem. Thus, the "Min To Front" problem is interesting because students already solved part of it (the find min part) in the previous lesson.

In the CSP Framework, almost every Essential Knowledge statement from Learning Objective **4.1.1 Develop an algorithm for implementation in a program** applies here.

The two points from the previous lesson carry over here and are also in the CSP Framework.

1. Different algorithms can be developed to solve the same problem
2. Different programs (or code) can be written to implement the same algorithm.

Furtermore the CSP Framework states:

- **4.1.1A Sequencing, selection, and iteration are building blocks of algorithms.**
- **4.1.2G Every algorithm can be constructed using only sequencing, selection, and iteration.**

The findMin problem and the other problems we solved with the Human Machine Language also have **sequencing**, **selection**, and **iteration**. Here's what they mean:

- **Selection**: also known as "branching" most commonly seen in if-statements -- The JUMP...IF command in the Human Machine

View on Code Studio

Objectives

Students will be able to:

- Develop an algorithm to solve a new problem with playing cards
- Express an algorithm in the Human Machine Language
- Identify Sequencing, Selection and Iteration in a program written the Human Machine Language
- Describe the properties of the Human Machine Language that make it a "low level" language.

Preparation

- [ ] Playing cards for students
- [ ] Decide which parts of the activity guide to print out
- [ ] **KEY Human Machine Language Pt 2, Min to Front**

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Human Machine Language - Part 2: Min To Front** - Activity Guide **Make a Copy**
- **You Should Learn to Program: Christian Genco at TEDxSMU** - Video

Vocabulary

- **Algorithm** - A precise sequence of instructions for processes that can be executed by a computer
- **Iterate** - To repeat in order to achieve, or get closer to, a desired goal.
- **Selection** - A generic term for a type of programming statement (usually an if-

Language is a form of selection. It gives us a way to compare two things (numbers) and take action **if** one thing was true.

- **Iteration**: also known as "looping" -- The JUMP command in the Human Machine Language allows us to move to a different point in the program and start executing from there. This allows us to re-use lines of code, and this is a form of iteration or looping.
- **Sequencing**: From the framework: "4.1.1B - Sequencing is the application of each step of an algorithm in the order in which the statements are given." Sequencing is so fundamental to programming it sometimes goes without saying. In our lesson, the sequencing is simply implied by the fact that we number the instructions with the intent to execute them in order.

Looking ahead, while sequencing seems obvious at first, it can trip up the novice programmer, because you must pay attention to how the state of the world changes with each line executed. In particular, this can cause some confusion when programming simple mathematics. For example, here is a series of instructions in some programming language:

```
x = 2
x = 5
x = x + 1
```

In mathematics this is an impossible (and meaningless) system of equations. But in programming, because it's sequenced, it simply means do one thing after the other. X gets the value 2. Then x gets the value 5. Then x gets the current value of x plus 1. When these 3 lines have completed executing x has the value 6.

Agenda

Getting Started (5 mins)

- **Conclusions from Find Min**
- **Creativity in Algorithms**

Activity (25 mins)

- **Adding SWAP to Human Machine Language**
- **Challenge: Min-to-Front**

Wrap Up (15-20 mins)

- **Review Min-to-Front Solutions**
- **Identify Sequencing, Selection, Iteration in Human Machine programs**
- **Discussion**
- **Video: You Should Learn to Program: Christian Genco at TEDxSMU - Video**

Extended Learning

Assessment

statement) that uses a Boolean condition to determine, or select, whether or not to run a certain block of statements.

- **Sequencing** - Putting commands in correct order so computers can read the commands.

Teaching Guide

Getting Started (5 mins)

Conclusions from Find Min

- If you didn't finish FindMin wrap up from the previous lesson, do it now.
- **Before moving on to the activity** students should all have **some** solution to FindMin written in the Human Machine Language.

Creativity in Algorithms

Transition to next challenge

One thing about algorithms is that once you know a few, and know how they work, you can combine them (or slightly modify them) to solve new problems. **Creativity** in algorithms comes from figuring out clever ways to solve problems by developing a process that could be executed by a machine.

Content Corner

This is almost verbatim a line from the CSP Framework.

- 4.1.1E Algorithms can be combined to make new algorithms.
- 4.1.1F Using existing correct algorithms as building blocks for constructing a new algorithm helps ensure the new algorithm is correct.

We study algorithms and care about them because in programming the techniques and mechanics of certain processes come up over and over and over again. So it's worth knowing a few so you don't have to reinvent the wheel.

For example, you just wrote an algorithm to find the smallest card in a row of cards. Is it hard to modify that exact same strategy to find the max card? (no).

Today we'll challenge you with a few more problems that will require you to get creative!

Activity (25 mins)

Adding SWAP to Human Machine Language

Distribute: Activity guide - **Human Machine Language - Part 2: Min To Front - Activity Guide**

- Put students into pairs

- Review the first page of the activity guide and the addition of the "swap" command.

Do the example program

- Give students a few minutes to trace the example program with their partner.

Review the example

Make sure that students understand the example before moving on to the challenge.

Here's what the example program does:

- END STATE: the order of the cards has been reversed
- It does this by first moving the right hand to the end of the list, then shifting each hand progressively toward the middle of the row, swapping cards each time.
- The program stops once the hands have crossed over each other (by checking if RHPos < LHPos)

Teaching Tip

You can either bring the whole class back together to review the example or you can check in with each pair as they finish the example and move them onto the challenge.

Challenge: Min-to-Front

The challenge is to find the min card and swap it to the front of the list, keeping the order of the rest of the cards the same.

As students work encourage them and challenge them to solve the problem in pieces.

Here are some suggestions you can make along the way:

- IDEA: Solve move-to-front first
  - Remember: "Algorithms can be combined to make new algorithms"
  - Students should know a solution to find min, so they can put that out of mind for a minute.
  - So, start by assuming that you've found the min card, and writing a procedure to move some card to the front of the list, by swapping.
  - Once you've got that you can tack the two together
- IDEA: Don't be afraid to invent a completely new algorithm
  - get creative - they might need or want to invent a whole new algorithm

Wrap Up (15-20 mins)

Review Min-to-Front Solutions

Put groups together to have students compare Min-to-Front solutions

- You can also have groups **trade algorithms** to test out each others' solutions

If necessary, show or reveal one or more of the solutions from the answer key.

- Make sure students see that it is possible to design a program to do Min-to-Front with the human machine language.

Teaching Tip

If students did some of the extra challenges (e.g. sorting, partition, etc.) you might give them an opportunity to showcase it for the class. Alternatively, if two or more people attempted a particular challenge you can put them together to compare solutions.

Identify Sequencing, Selection, Iteration in Human Machine programs

Remarks

> The CSP Framework states:
>
> - **4.1.1A Sequencing, selection, and iteration are building blocks of algorithms.**
> - **4.1.2G Every algorithm can be constructed using only sequencing, selection, and iteration.**
>
> If these statements are true then we should be able to identify these elements of sequencing, selection and iteration in our Find-Min and Min-to-Front algorithms.
>
> I'll give you a quick definition of each and you tell me if or where we saw it in our Human Machine Language programs...

Prompt:

- **"4.1.1B Sequencing is the application of each step of an algorithm in the order in which the statements are given." -- Does our human machine language have sequencing?**
  - Sequencing is so fundamental to programming it sometimes goes without saying. In our lesson, the sequencing is simply implied by the fact that we number the instructions with the intent to execute them in order.

Prompt:

- **"4.1.1C Selection uses a [true-false] condition to determine which of two parts of an algorithm is used." -- Where did we see "selection" in our human machine language programs?**
  - The JUMP…IF command in the Human Machine Language is a form of selection. It gives us a way to compare two things (numbers) and take action if the comparison is true, or simply proceed with the sequence if false. NOTE: Selection is also known as “branching” most commonly seen in if-statements in programs.

Prompt:

- **"4.1.1D Iteration is the repetition of part of an algorithm until a condition is met or for a specified number of times." -- Where did we see iteration in our human machine language programs?**
  - The JUMP command (as well as JUMP...IF) in the Human Machine Language allows us to move to a different point in the program and start executing from there. This allows us to re-use lines of code, and this is a form of iteration or looping.
  - NOTE: Iteration is also known as “looping” in most programming languages.

Discussion

After reviewing solutions there are a few points to make to wrap up this foray in to algorithms:

Algorithms can be combined to make new algorithms

- This is an important **essential knowledge statement** directly from the CSP Framework. Students should see the connection between the the Find-Min problem and the Min-to-Front problem.

Low-Level languages exist

- Most programming languages that you use in every day life are simply higher level, perhaps easier-to-read commands that are translated into more primitive machine commands. So-called "low level" languages are the most basic, primitive, set of commands to control a computer. The Human Machine Language is similar to something called **Assembly Language** (see graphic)
- From the CSP Framework: **2.2.3C Code in a programming language is often translated into code in another (lower level) language to be executed on a computer.**
- The Human Machine Language is a "low level" language because the commands are very primitive and tie directly specific functions of the "human machine".

Content Corner

Here is an example of a simple program written in IBM Assembly Language.

```
; Example of IBM PC assembly language
; Accepts a number in register AX;
; subtracts 32 if it is in the range 97-122;
; otherwise leaves it unchanged.

SUB32  PROC         ; procedure begins here
       CMP  AX,97   ; compare AX to 97
       JL   DONE    ; if less, jump to DONE
       CMP  AX,122  ; compare AX to 122
       JG   DONE    ; if greater, jump to DONE
       SUB  AX,32   ; subtract 32 from AX
DONE:  RET          ; return to main program
SUB32  ENDP         ; procedure ends here
```

FIGURE 17. Assembly language

Read more about **Assembly Language on Wikipedia**.

Teaching Tip

Students don't really need to know many specifics about low level languages. The point should be made here in order to refer back to it later when students are programming in a high level language (like JavaScript). A high level language is more abstract, provides more functionality to make it faster to write and reason about programs, but ultimately that code is translated down into low-level, primitive machine instructions for the computer to execute.

Video: You Should Learn to Program: Christian Genco at TEDxSMU - Video

NOTE: this video is 10 minutes long. If possible, and to save time, you might ask students to watch it outside of class.

Remarks

> Learning to program is **really** learning how to think in terms of algorithms and processes. And it can be really fun and addicting. It also can make you feel like you have magical powers.
>
> In the next lesson we'll start writing programs that a real machine (not a human machine) will execute. But since programming is really about thinking and problem solving your "human machine" skills will come in handy - reasoning about program is also about reasoning about what a computer can and cannot do, and what the given language you're using let's you and doesn't let you do.
>
> If you didn't want to learn how to program before, perhaps this video will change your mind...
>
> **Show video: You Should Learn to Program: Christian Genco at TEDxSMU - Video**

Extended Learning

Try out Human Machine Language on a computer: Try out this **Plugged Version of the Human Machine Language** designed by CS Principles teacher Brian Dahelm. Consider using this tool to plug in some of the activities in this lesson.

Assessment

Write a human machine language program that: **Repeatedly shifts the left hand to the right until it finds a 5 or 6** The program should stop when the left hand is at (or past) the end of the list, or it finds a 5, or it finds a 6.

Standards Alignment

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **4.1** - Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **5.2** - People write programs to execute algorithms.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes, **contact us.**

Lesson 4: Using Simple Commands

Turtle Programming | App Lab

Overview

This lesson is a student's first exposure to programming in App Lab. The lesson begins with a quick reflection prompt. Then students are introduced to the practice of pair programming before beginning to program. For this lesson the students' view is limited to only a very few simple “turtle” commands to draw graphics on the screen. After a few warm up exercises, using only combinations of four drawing commands, students must figure out the most “efficient” way to draw an image of a 3x3 grid. The lesson concludes with a sense-making discussion about the meaning of efficiency in programming and the reason behind beginning with such a limited set of programming tools.

Purpose

The overarching purpose of this lesson is to start students on their programming journey on the right foot. It should feel like a natural extension of the collaborative problem-solving they've done so far in the course.

One way this is done is by introducing pair programming, a pedagogy with demonstrated benefits on student learning and attitudes. While you shouldn't use pair programming every time students program, introducing the practice early means you can call back to it when appropriate through the rest of the unit.

This lesson also aims to frame programming as a creative problem-solving discipline. One way this is done is by limiting students to a small set of commands, namely `moveForward`, `turnLeft`, `penUp` and `penDown`. These constraints move the focus from memorizing commands and syntax to creatively examining problems, iterating on ideas, and collaborating with peers. The closing remarks highlight the value of these constraints and the authenticity of needing to program with seemingly limited commands.

Agenda

Getting Started (10 mins)

- **Transitioning to Programming**
- **Introduce Pair Programming**

Activity (30 mins)

- **Move to App Lab**

Wrap-up (15 mins)

- **Discussion: What does “efficiency” mean when programming?**
- **What's the point? Why constrain to 4 commands**

View on Code Studio

Objectives

Students will be able to:

- Use App Lab to write programs that create simple drawings with “turtle graphics.”
- Create a program with a partner following the pair programming model
- Explain different ways one might measure the efficiency of a program.

Preparation

- [ ] Practice the levels yourself so you're familiar with the challenges.
- [ ] Familiarize yourself with opening remarks and comments for background knowledge.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Tutorial - Turtle Programming (In Code Studio)** - Video (**download**)

Vocabulary

- **Pair Programming** - A method of programming in which two programmers write code using a single computer. One programmer in the "driver" role uses the mouse and keyboard to actually write the code while a second acts as a "navigator", keeping track of the big picture, catching errors, and making suggestions. Programmers switch roles frequently and communicate throughout the process.
- **Turtle Programming** - a classic method for learning programming with commands to control movement and drawing of an on-screen robot called a "turtle". The turtle

Extended Learning Assessment

hearkens back to early implementations in which children programmed a physical robot whose dome-like shape was reminiscent of a turtle.

Introduced Code

- moveForward
- turnLeft
- penUp
- penDown

Teaching Guide

Getting Started (10 mins)

Transitioning to Programming

Prompt: Think back over the human machine language lessons we just completed. Write down some advice or encouragement you'd give to students in next year's class before they complete those same lessons. What attitudes or skills will be important?

Discuss: Have students silently journal responses. Then have them share with partners. Then discuss with the whole class. You may want to record their responses on the board.

Discussion

Goal: This should be a quick discussion aimed at building confidence. Many students are likely excited to program, but others may be apprehensive. This prompt and the following quick words should be used to quickly reassure students they've already practiced many skills necessary to learn programming. They're ready for this!

Obviously you can't control what responses students will give, but once you've heard a few students advise students to stick with it, or check their work carefully, etc. you're likely able to summarize and move on.

Remarks

> These are some great pieces of advice. Now I'm going to reveal that this advice isn't really for next year's students, but for all of you! Today we're going to begin programming on the computer, and while that's obviously different from the human machine language activities we just completed, a lot will be the same. You'll need to stick with it, pay attention to detail, and collaborate with others. In fact, before we jump in there's one new collaboration skill I want to teach you all.

Introduce Pair Programming

Remarks

> Today we'll be using a practice called **pair programming**. Pair progamming helps people make better programs by working together. The main idea is that two programmers use a single computer to collaboratively develop code. There's a few rules to make sure this system works effectively.

Review: Introduce the rules of pair programming below.

- There is only one computer and two programmers.
- Assign one student "driver" and the other "navigator". Don't worry, you'll switch often.
- The driver is the only one to touch the keyboard/mouse.
- The navigator should look for problems in the code and keep track of the high-level plan.
- Both driver and navigator should be communicating constantly by talking out loud and sharing ideas.
- Driver and navigator roles must switch when the teacher indicates, typically every 3-5 minutes.

Teaching Tip

Pair Programming: Pair programming is practiced in education and in industry. Students who pair program are more confident in their abilities and are more likely to continue to study computer science. This practice is most effective if it is introduced early and the rules for switching partners are enforced by the teacher. You can read more about Pair Programming in the **Computer Science Principles Curriculum Guide** on page 8.

Enforcing Timing: A particularly important role of the teacher of this lesson is introducing the logistics of pair programming. Enforce students trading who is driver or navigator every few minutes (3 minutes). Use a timer and actually force pairs to show you that they are switching roles.

When to Use Pair Programming: You don't need to always pair program. You may even opt to stop pair programming after 15 minutes or so of this lesson. Introducing the practice early and reinforcing its value will ensure it's a practice you can return to throughout this unit.

Activity (30 mins)

Move to App Lab

Group: Place students in pairs. If you need to make one group of three you may do so and assign two members at each time to be "navigators".

Remarks

> To get you situated in App Lab there's a quick video. Once everyone is through that we'll get started. I'll keep track of time and remind you to switch who is driver and navigator every 3 minutes or so. Make sure you're ready to switch by talking out loud and sharing ideas.

Code Studio levels

- Lesson Overview
- **Student Overview**

View on Code Studio

- Video: Turtle Programming
- **Student Overview**

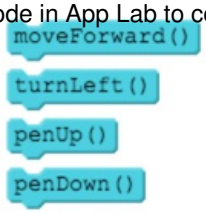
View on Code Studio

- Levels
- **3**
- **4**

View on Code Studio

Student Instructions

Given only 4 turtle commands, write code in App Lab to complete the drawings in each of the challenges below. Your goal is to write code that solves the problem as efficiently as possible using only the commands provided.



What does “efficient” mean? In computer science it can actually mean several different things depending on what you’re talking about. Work on these challenges, try to solve them in a way that you think is best and we’ll ask about efficiency at the end.

Teaching

The constraint of working with only 4 commands is important. (See: “What’s the Point?” below.) If students come to the class with prior knowledge of programming they might try to go outside these constraints to reduce the number of lines of code.

If you see students using commands beyond the 4 provided - or using parameters with those commands like: moveForward(150) , you should tell them that they are not solving the problem; the problem is to create the images using only the 4 commands provided.

Do This:

Draw a 1 x 1 square to the front and left of the turtle as efficiently as possible. The program should stop with turtle in its original position, facing its original direction.

Once you think you have a solution, click the Finish button - it will appear once you run the program.

Student Instructions

View on Code Studio

Do This:

Teaching

The insight here is that we can turn right by turning left 3 times.

This time, draw a 1 x 1 square to the **front and right** of the turtle as efficiently as possible. Notice that there is no command to turn right! You need to figure out how to draw this by only turning left and moving forward. The program should stop with turtle in its original position, facing its original direction.

When you're done click the Finish button to move onto the next problem.

Wrap-up (15 mins)

Discussion: What does “efficiency” mean when programming?

Compare Solutions and Discuss: Each pair of students should get together with another pair to compare their solutions for each of the three problems to determine which is the most efficient. Once they have shared and compared their solutions they can consider the questions below.

- What strategies or reasoning did you use to identify possible solutions?
- Is the solution that you or another group found the most efficient? How do you know?

Prompt: What is the “most efficient” way to program the solution for the 3x3 grid?

Discuss: Have students share at their tables before moving to a full class discussion.

Prompts:

- Today’s activity challenged us to find the most efficient solution to a problem. We care about efficiency when we don’t want to waste something valuable, like money, time, or space. We measured the efficiency of our programs in terms of lines of code, but there are other ways to think about efficiency when talking about code or programs that run on a computer. **When we try to create efficient programs, what other valuable resources might we be concerned about conserving?**

Discussion

The point of this discussion is to highlight the fact that "efficiency" can mean different things depending on the context.

Potential answers to this question can include the time they take to run on a computer, the time or number of people they require to create, the number of resources they require to run, etc. Students will likely come up with others.

Don’t fixate on any one solution and encourage students to come up with the most comprehensive list possible.

- List the things that students come up with somewhere that can be read by the class. These can be used to motivate portions of the next and future lessons.

What’s the point? Why constrain to 4 commands

Remarks

Why did we constrain you to only working with 4 possible commands? We begin programming with only a few commands for **a few main reasons:**

1. What you did today is what you always do when programming: use a language to express a solution to a problem. **Collaborative problem-solving** skills are important factors in programming. Being able to get together with other people to talk about and make reasoned arguments about the best ways to code things is what professionals do.
2. Learning to program isn’t just a matter of memorizing commands. The art of programming - **the creative part** - is always about understanding how to use the features of a programming language to solve a problem.
3. Whether you know 4 commands of a language or hundreds, **you will always be constrained by the programming language.** The reason you need to learn how to program is because there isn’t a command to do every little thing; you have to understand the set of things that a programming language can do, and then use your creativity and problem-solving skills to get the computer to do what you want.
4. In this class, we will repeatedly run into the challenges we encountered today. Even seemingly simple problems will require us to creatively apply the tools we have available: the commands or instructions provided by the programming language.
5. Multiple approaches are always possible to solve a problem. Furthermore, we may never be sure we have found the

Teaching Tip

- There are a lot of points in this “What’s the Point?” section. They all go together and they are not all points you have to make in this lesson right now.
- You should keep these big ideas in your back pocket, because you may want to refer back them in the future.
- The short version:
  - **Creativity** in programming comes from the fact that all programming languages have constraints; your job is to figure out how to use the language to get the computer to do what you want it to do or to solve a problem.
  - **Collaboration** is a huge help when solving problems with programming. It’s helpful to work through problems with a partner.

most efficient one. We may not even agree about how we should measure efficiency.
6. As we saw today, even when designing programs using four commands to draw with a turtle, the solutions are not always easy. We will definitely need to keep practicing as we start tackling more complex challenges.

Extended Learning

Ask students to propose ways they could prove that their solution to the 3x3 grid problem is the most efficient. **Note:** An actual proof is very challenging.

Assessment

Criteria for 3-by-3 Grid:

- If you want to assess students' programs (especially 3x3 grid) you can see their work in the answer viewer and you might look for the following criteria:
  - The program draws the desired figure.
  - The turtle is returned to its starting location and is pointed in the correct direction.
  - The figure is located in the correct position, relative to the starting position of the turtle.
  - The program only makes use of the "building block" commands provided.
  - The program is reasonably efficient in its execution.

Online Questions: A set of assessment / reflection questions is available on Code Studio.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **5.1** - Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **5.2** - People write programs to execute algorithms.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 5: Creating Functions

Turtle Programming | App Lab

Overview

In this lesson, students learn to define and call procedures (in JavaScript, procedures are called “functions”) in order to create and give a name to a group of commands for easy and repeated use in their code. They will be introduced to functions as a form of abstraction that enables them to write code in larger, more logical chunks and focus on what something does, rather than how it does it. As they explore the use of functions through a sequence of activities in App Lab, they will be asked to think about where they see the need for functions and how these functions can make their code clearer or more concise.

At the end of the lesson students review the concept of abstraction and are introduced to elements of the Create PT in preparation for the Practice PT at the end of the unit.

Purpose

Programming languages must necessarily define the meaning of a set of commands which are generally useful and applicable. In order to extend their functionality, nearly all programming languages provide a means for defining and calling new commands which programmers design to fit the needs of the problems they are trying to solve. Defining functions is an example of how computer scientists use abstraction to solve problems. A programmer will design a solution to a small, well-defined portion of the task and then give a name to the associated code. Whenever that problem arises again, the programmer can invoke the new function by name, without having to solve the problem again. This ability to refer to complex functionality by simple, meaningful names allows for programs to be written in more intuitive ways that reflect the relationships between different blocks of code.

Agenda

Getting Started (5 min)

- **Discussion - The Need for Functions**

Activity (35 min)

- **Programming with Functions**

Wrap-up (15 min)

- **Introduce Create PT (10 mins)**
- **Reflection on abstraction and managing complexity (5 mins)**

Assessment

View on Code Studio

Objectives

Students will be able to:

- Recognize functions in programs as a form of abstraction.
- Write a program that solves a turtle drawing problem using multiple levels of abstraction (i.e. functions that call other functions within your code).
- Explain why and how functions can make code easier to read and maintain.
- Define and call simple functions that solve turtle drawing tasks.

Preparation

- [ ] Review student-facing instructions in Code Studio

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **College Board - Assessment Overview and Performance Task Directions for Students**
- **Intro to Functions Tutorial** - Video (**download**)

Vocabulary

- **Abstraction** - a simplified representation of something more complex. Abstractions allow you to hide details to help you manage complexity, focus on relevant concepts, and reason about problems at a higher level.
- **Function** - A named group of programming instructions. Functions are reusable abstractions that reduce the complexity of writing and maintaining programs.

Assessing Programs Questions

Extended Learning

Share and Compare

Keeping Going!

Introduced Code

- function myFunction() { // function body, including optional "return" command. }
- Call a function

Teaching Guide

Getting Started (5 min)

Discussion - The Need for Functions

Prompt: In the previous lesson we created simple turtle drawings using only four commands. At some point you probably wished that more commands were available to you. Describe a command you wanted to be able to use and explain why you wanted to use it.

Discuss: Provide students time to discuss their ideas with their neighbors before sharing with the class. This can be done fairly quickly but insist that students provide both the command they would have wanted and the situation in which they wanted to use it.

These will be addressed in a later lesson. Remind them they are thinking of commands to help with the last lesson's challenge.

Discussion

Goal: Students have been solving turtle challenges with a limited set of commands. Recognize their desire to have access to more commands to solve these challenges. Motivate the fact they can create these new commands themselves, using the original turtle commands as building blocks.

Responses might include:

- A command that turns the turtle to the right
- A command that moves forward more than one space
- A command that creates a repeated pattern or shape
- Students may also just wish for different colors, curved lines, etc.

Remarks

> Programming languages will always have some commands that are already defined, but there will be many instances when the exact command we want isn't available. Today we're going to start exploring a powerful feature of most programming languages that will allow us to overcome this issue and create commands of our own.

Activity (35 min)

Programming with Functions

Group: Place students in pairs. Consider using pair programming for this activity. If each student will work on their own computer have them check in with their partner throughout the activity.

Transition: Have students log in to Code.org

Code Studio levels

Lesson Overview

Teacher Overview | Student Overview

Either as a class or individually, students should watch a video introducing the way that functions are defined and called in JavaScript. You may wish to take a minute following the video to address any questions that may have developed from watching the video.

View on Code Studio

Defining and Calling Functions

2 *(click tabs to see student view)*

Using Functions With Turtle Commands

3 | 4 | 5

(click tabs to see student view)

CSP Abstraction in Programming 2_2018

Teacher Overview

Student Overview

How to use this level

This level has a lot of text. Ways you might use it / incorporate it into your class:

- Assign as reading for students the day before
- Have students stop at this level during the normal progression and read as a group - discuss key points.
- Read and summarize for your students
- Make note of it as a reference for students that explains "abstraction"
- Use in conjunction with a preview of the AP Create Performance Task

View on Code Studio

Using Functions With Turtle Commands

7 8 9

(click tabs to see student view)

Challenge: Draw a Diamond Using Functions

10

(click tabs to see student view)

Check Your Understanding

11 12 13 14

(click tabs to see student view)

Wrap-up (15 min)

Introduce Create PT (10 mins)

Distribute: Give students digital or printed copies of **College Board - Assessment Overview and Performance Task Directions for Students**. We will review pages 9-11 which introduces the Create PT Components (Digital copy linked to from student resource section for this lesson on code studio).

Remarks

> At the end of this unit we will be doing a Practice Create PT that mimics most components of the AP Create PT. We're going to quickly review those components now, but we'll have opportunities to review and practice them in the next few lessons as well. For right now you don't need to understand all the details, just the big picture.

Review: Quick skim this document with the class, touching on the following points.

- Page 9: The Create PT has 3 major components, 1. video, 2. written responses 3. program code. Students will skip the video this time as well as written response 2c.
- Pages 10-11: Skim the submission requirements and give students time to read prompts 2a - 2d.
- Highlight prompt 2d which references abstraction, the theme of this lesson and a focus of this unit.

Create PT discussion

Goals:

- Foreshadow the Practice PT at the end of the unit that has been designed to mimic most of the elements of the real AP Create PT
- Students are aware of the major components they have to submit for the AP Create PT (1. annotated PDF of program code 2. wirtten responses 3. video)
- Students understand that one of the key elements of the Create PT is identifying and responding to written prompts about an abstraction that they created to manage complexity.

Keep it short: Avoid making this a 45-minute deep-dive review of the AP Create PT and all its components - a surface level understanding is all that is needed to proceed. Practice questions and direct references to College Board materials are included in each of the next 5 lessons. This is a kickoff discussion, and you will have many opportunities to prepare students for the PT in later lessons.

Discuss: Respond to any questions students share. Don't lose too much time here. You'll have many opportunities to review the Create PT in later lessons.

Reflection on abstraction and managing complexity (5 mins)

Display the following questions somewhere they can be seen and ask students to write short responses before discussing with a neighbor. Once all pairs have had a chance to talk, bring the whole class back together to clarify points and share ideas.

Remarks

> Let's start now making connections between writing functions and "developing abstractions" that help us when writing to programs

Prompt 1: List the benefits of being able to define and call functions in a program. Who specifically gets to enjoy those benefits?

Prompt 2: How is the use of a function an example of abstraction?

Reflection on Abstraction

Benefits of functions -- Possible responses:

- programs are easier to read and write
- functions remove the need to repeat large chunks of code
- functions break the program into logical chunks
- Functions are a way to give concise human-readable names to complex sections of code
- Important to note: functions do not make the program easier (or harder) for the computer to run or understand -- it must still run all of the same commands as before.

Functions are examples of Abstraction -- Possible responses:

- Abstractions simplify the representation of something more complex. A function gives a simple name to what might be a complex chunk of code.
- Writing a function is a way to compartmentalize an idea
- Abstraction helps manage complexity - so do functions because they let you break a problem into smaller parts.
- Writing a function is a way to solve and then ignore lower-level, nitty-gritty problems in your program so you can focus on higher level problems.

Assessment

Assessing Programs

If you wish, you may assess the final version of the

- The program draws the diamond.
- The program defines four functions: right(), drawStep(), drawSide(), and drawDiamond(). The names are less important than the existence of four functions with this functionality.
- The program makes a single call to drawDiamond().
- The program looks clean and organized.

Questions

1. Check the two items that are true statements about functions:
   - Meaningful function names help people better understand programs.
   - Meaningful function names help computers better understand programs.
   - Functions in programming are useful mathematical tools for doing complex computations.
   - Functions in programming are named groupings of programming instructions.
2. In your own words explain at least one reason why programming languages have functions.

Extended Learning

Share and Compare

If students complete the diamond challenge early, ask them to compare their solution with another group. They should then try to create the version of the program that they believe is best with an associated justification. Challenge students to think about potential tradeoffs between efficiency and readability.

Keeping Going!

Groups that finish early can continue working with their functions to make more complex figures. Ask groups to continue building functions they think are generally useful for drawing.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **5.3** - Programming is facilitated by appropriate abstractions.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **CS** - Computing Systems

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 6: Functions and Top-Down Design

Turtle Programming | App Lab

Overview

This lesson presents a top-down problem-solving strategy for designing solutions to programming problems. Students use a worksheet to learn about top-down design, and then on paper, design a solution to a new turtle drawing challenge with a partner. Having practiced this approach on paper and in code, students will be re-presented with the 3x3 square challenge from an earlier lesson and asked to improve upon their old solution by designing multiple layers of functions.

Purpose

The main purpose here is to familiarize students with the concept of developing "procedural abstractions." One of the main learning objectives in the CSP framework is: "2.2.1 Develop an abstraction when writing a program...". Furthermore the **AP Performance Task: Create** requires students to "develop an abstraction to manage complexity of your program." For all intents and purposes, developing abstractions starts with writing reusable functions (or procedures) in your code that encapsulate and give a name to multi-line segments of code that other parts of your code calls upon.

A technique for deciding what functions you should write is to look at the problem with a "top-down design" perspective. The process of creating software begins long before the first lines of code are written. Breaking a problem down into layers of sub-tasks, and writing well-named functions that solve those tasks is a creative act of abstraction. It also leads to good code that is more efficient, easier to read, and therefore easier to debug when problems arise.

In professional settings, teams of people first identify the problems and sub-problems the particular software will be addressing and how it will be used. This approach to designing software is critical when facing large-scale programming tasks. Once the problem is well understood, it can be broken into parts that teams or individual programmers can begin to work on solving at the same time. Full software systems take advantage of the power of abstraction; each programmer in a team can write code, assuming the subproblems will be solved and written by other teammates.

Agenda

Getting Started (5 min)

What Does Efficiency Mean?

Activity

View on Code Studio

Objectives

Students will be able to:

- Write a complete program with functions that solve sub-tasks of a larger programming task.
- Explain how functions are an example of abstraction.
- Use a “top-down” problem-solving approach to identify sub-tasks of a larger programming task.

Preparation

- [ ] Review this **explanation of Top-Down Design**

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Top-Down Design** - Worksheet
  Make a Copy

Vocabulary

- **Abstraction** - a simplified representation of something more complex. Abstractions allow you to hide details to help you manage complexity, focus on relevant concepts, and reason about problems at a higher level.
- **Function** - A named group of programming instructions. Functions are reusable abstractions that reduce the complexity of writing and maintaining programs.
- **Top Down Design** - a problem solving approach (also known as stepwise design) in which you break down a system to gain insight into the sub-systems that make it up.

- **Distribute Worksheet**
- **Transition to Code Studio**

Wrap-up

- **Some points about functions and abstraction:**

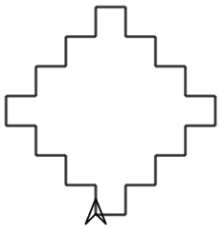
Assessment

- **Assessing Programs**
- **Questions**

Teaching Guide

Getting Started (5 min)

What Does Efficiency Mean?



Recall:

In the previous lesson we wrote a program that used layers of functions (functions that called other functions) to get the turtle to draw a diamond-shaped figure.

Prompt:

"Imagine that you have two programs that drew the diamond-shaped figure. One program uses functions as we did in the previous lesson. The other doesn't use functions; it's just a long sequence of the turtle's primitive commands. Which program is more efficient? Make an argument for why one is more efficient than the other."

Discuss:

Have students briefly share their arguments for one program over the other and steer the conversation to these points… "Efficient" can mean several different things here:

- It could mean the total number of primitive operations performed by the turtle.
- It could mean number of lines of code.
- It could hinge on the ability to reuse code within your own code.
- It could be about the speed and clarity with which you can write the program.

All of these are valid arguments for a student to make.

Transition: Efficiency is an interesting thing to think about, but functions also introduce the ability to leverage the power of abstraction: when we write a function, it solves a small piece of a bigger problem. Having the small problem solved allows us to ignore its details and focus on bigger problems or tasks.

Today we'll get more practice with thinking about how to break problems down into useful functions.

Activity

Distribute Worksheet

Distribute the **Top-Down Design - Worksheet**

- Students should work in pairs.
- Read the first page of the worksheet that describes the top-down problem solving process.
- Design a solution to the problem on the second page by writing down the functions they would write to solve the problem.
- After a pair has come up with a solution on paper, have them compare with another group to see similarities and differences.

Transition to Code Studio

Here is a quick reference guide for what students will see in Code Studio.

Teaching Tip

Comparing solutions on paper

The point of having students compare top-down designs is mostly just to see that people think about problems differently. Pairs should move to Code Studio after working out a solution on paper.

More practice with top down design?

If you want to give students more practice with top-down design you can make up almost any geometric shape that has some patterns or repetitions in it. Virtually any symmetrical figure you can create with the turtle would have some elements worth breaking into functions. Here's a trick to make your own: make a function that has the turtle draw **something**. Then call that function a few times, perhaps turning or moving between each call. Use the resulting figure as an interesting exercise.

Evidence of good design

An interesting thing to look at is the name of the top-level function students come up with, which should be a description of what they think they are drawing.

Guide students to come up with descriptive function names that give insight to their thinking. A high-level function named something like `drawThing` is not that great. Something like `snowflake` or `crosshairs` or `antenna` is better because it lets the reader see what the programmer is thinking about.

Code Studio levels

Lesson Overview

Student Overview

Using Functions with Turtle Commands

2 | 3 *(click tabs to see student view)*

AP Practice Response - Design Process

4 *(click tabs to see student view)*

Check Your Understanding

5 | 6 | 7 *(click tabs to see student view)*

Wrap-up

Some points about functions and abstraction:

- When we layer functions - with functions that call other functions - we are creating layers of abstraction.
- In programming, writing functions helps us create layers of abstraction that allow us to design and create increasingly complex systems.
- We've seen layers of abstraction before in the design of Internet protocols, or in the binary encoding of information.
- Solving a fundamental piece of a problem that can be reliably reused in a different context frees us to think about more complex problems because we can trust the piece that has been solved and don't have to worry about its details.
- Solving small problems - like how to send a single bit from one place to another - allows us to think about bigger

problems, like sending numbers, or text, or images, to multiple people, over networks, in packets...etc., etc., etc.

Prompt: "Where else in your life have you seen layers of abstraction? Connect the idea of layers of abstraction to some other activity."

- There are many possible connections to make since almost any task in life can be broken down ad-infinitum. In the video about functions from the previous lesson, Chris Bosh makes a connection to basketball. Once you know how to dribble and shoot, then you learn some moves to do, and once you know that, you run plays that rely on the fact that those fundamental elements (or the problems) have been solved.

Assessment

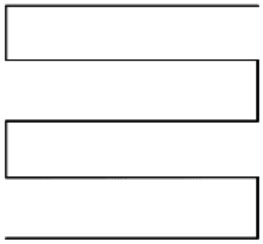
Assessing Programs

If you wish, you may assess the two programs written in Code Studio; the criteria are the same for both the “snowflake” program and the new version of the 3x3 grid:

- The program draws the figure correctly.
- The program defines multiple layers of functions.
- The functions defined have descriptive and meaningful names.
- The program is “kicked off” with a single call to a function that makes calls to subsequent functions.
- The program looks clean and organized.

Questions

1. Consider the figure below. Use top-down thinking to design a solution to the problem. In the space provided write a list of just the names of the functions that you would write in a program that draws this figure. (Assume that the long line segments are 6 turtle moves long.)



2. Which of the following statements about writing functions and Top-Down Design is NOT true?
    - Writing functions helps manage complexity in a program.
    - Top-Down Design leads to programs which feature multiple layers of abstraction.
    - Two programmers solving the same problem using Top-Down Design should arrive at identical programs.
    - Top-Down Design relies upon identifying subproblems of a larger problem.
    - Top-Down Design assists in identifying the layers of functions that will be used to solve a programming problem.
3. In the Create Performance Task you will be asked to identify an abstraction in your program and explain how it helps manage the complexity of the program. Functions are a form of abstraction. Pick a function you wrote in your solution to the 3x3 square problem and explain how it helps manage the complexity of your program.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **5.1** - Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **5.3** - Programming is facilitated by appropriate abstractions.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 7: APIs and Using Functions with Parameters

App Lab

Overview

Students will learn to read App Lab's API documentation and will use functions that accept parameters in order to complete a series of drawing puzzles which require them to make use of the App Lab API documentation to learn new drawing commands. Many of these commands will require the use of parameters. The final challenge asks students to design a personal monogram making use of the commands they learned during the lesson.

Purpose

An API is a reference guide which catalogs and explains the functionality of a programming language. If a programmer develops the practice of referencing an API, she can make full use of that functionality without undergoing the tedium of memorizing every detail of the language. In today's lesson, students will need to read through the API in order to find and understand new commands for moving the turtle, selecting colors, and drawing different-sized dots and lines on the screen. Students should not necessarily understand every command in the drawing API in detail, but they should be familiar with referencing the API as a standard part of the process of writing a program. This will also be the first time students are given access to drawing functions that take parameters (e.g., moveForward(40) vs. moveForward()).

Agenda

Getting Started (5 mins)

Activity (40 mins)

- **Learning the API and Using Functions with Parameters**

Wrap Up (5 mins)

- **Reviewing Vocabulary**
- **Share Your Images**

Assessment

Extended Learning

View on Code Studio

Objectives

Students will be able to:

- Use parameters to provide different values as input to procedures when they are called in a program.
- Use API documentation to assist in writing programs.
- Define an API as the set of commands made available by a programming language.

Preparation

- [ ] Review functions in the "Turtle toolbox to assist in debugging code

Vocabulary

- **API** - a collection of commands made available to a programmer
- **Documentation** - a description of the behavior of a command, function, library, API, etc.
- **Hexadecimal** - A base-16 number system that uses sixteen distinct symbols 0-9 and A-F to represent numbers from 0 to 15.
- **Library** - a collection of commands / functions, typically with a shared purpose
- **Parameter** - An extra piece of information passed to a function to customize it for a specific need

Introduced Code

- moveTo
- turnTo
- arcRight
- arcLeft
- penWidth
- penColor
- penRGB

- Call a function with parameters

Teaching Guide

Getting Started (5 mins)

Remarks

> So far in this unit we have been exploring programming by drawing turtle art with only a few commands. It will probably not surprise you to learn that there are many more commands included in most programming languages, including the version of JavaScript included in App Lab. In fact, most programming languages include hundreds if not thousands of commands.

Discussion

Introduce the fact that most programming languages, including the version of JavaScript included in App Lab, contain many commands. Motivate the need for documentation to help programmers learn and recall how to use these commands.

Thinking Prompt:

"Do you think programmers memorize all of the commands in a programming language? If not, how is anyone ever able to use an entire programming language?"

Discuss: Provide students time to discuss their ideas with their neighbors before sharing with the class. Responses might include:

- Programmers ask each other for help.
- There are reference guides online or in print to help programmers use a language.
- Programmers read old code in the language to learn how to write new code.

Teaching Tip

What Makes a Programmer?: This discussion is a great opportunity to emphasize that programming is not some innate talent. Every programmer has to learn by seeing examples of a language being used and reading documentation. In fact, even professional programmers will frequently reference documentation while designing software. Becoming a good programmer is much less about memorizing a language and more about learning habits of mind and patterns that allow you to use a language (including its documentation) effectively!

Transitional Remarks

> Programmers weren't born knowing how a programming language works, and, like you and me, they don't have perfect memories. Instead they rely on written documentation to help them learn new features of a language and recall how it works. Today we are going to be exploring how useful documentation can be when learning a programming language or just writing software.

Activity (40 mins)

Learning the API and Using Functions with Parameters

Remarks

> At end of the day we hope to be much more talented turtle artists, but we're also going to be learning another important skill: reading the documentation of a programming language.
>
> Before you ask me or a classmate for help today, I want you to read through the documentation, try the examples, talk with friends, and then talk to me. It may be slower going today, but in the long term it will make you much more confident programmers.

Code Studio levels

Lesson Overview

Teacher Overview | Student Overview

For Teachers Only A new version of the `moveForward` command is introduced: `moveForward(pixels)`. Students use it to move the turtle to the edge of the screen in one command.

View on Code Studio

Encourage students to read the explanation of the term parameter and not just skip it. It's an important term to know.

Terminology: Parameter is introduced. Students are shown parameterized versions of basic turtle move commands and asked to draw a triangle.

Introduction to Parameters

2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

(click tabs to see student view)

Make Your Own Turtle Drawing

10

(click tabs to see student view)

AP Practice Response - Score a student response

11

(click tabs to see student view)

Check Your Understanding

12 | 13 | 14 | 15

(click tabs to see student view)

Wrap Up (5 mins)

Reviewing Vocabulary

The level progression for today's lesson includes many important vocabulary words. While these levels attempt to introduce these words in the context of using them, take a moment at the conclusion of class to review the words covered and ensure students are comfortable using them.

- **Parameter:** accepts a value to be passed to a function, typically affecting the behavior of that function (e.g., changing the distance the moveForward() command moves the turtle)
- **Library:** a collection of commands / functions, typically with a shared purpose (e.g., a library of functions for manipulating the turtle)
- **API:** application program interface, the full set of commands included in a programming language (e.g., every command made available by App Lab)
- **Documentation:** a description of the behavior of a function, library, API, etc.

Share Your Images

If there is time remaining, provide students an opportunity to share images they created using the full turtle library.

Assessment

Questions:

1. Multiple Choice: What is an API?

- Abstract Programming Inheritance: The idea that abstractions in languages get “passed down” in newer versions
- Artificial Parameter Intelligence: The idea that function parameters should be intelligent enough to “know” what you want as a programmer
- Application Program Interface: A well-documented library of functions provided in a programming language that helps to simplify complex programming tasks.
- Abstract Parameter Interface: A high-level description of the parameters a function accepts

2. Multiple Choice: What is a function parameter?
   - “para-meter” -- a measure of the distance between a function’s conception and implementation.
   - A way to give input to a function that controls how the function runs.
   - A collection of commands that can be used in a programming language.
   - Another name for the purpose of a function.
   - A named memory location.
3. Free Response: It is said that functions with parameters generalize the behavior of a more specific command. Explain what this sentence means to you using the difference between turnLeft() and turnLeft(angle).

Extended Learning

Design an image using the drawing commands reviewed today. Share the image (but not the code used to generate it) with a classmate and challenge him to recreate the image. How close can he get? Did you complete the task in the same way?

Standards Alignment

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **5.1** - Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **5.3** - Programming is facilitated by appropriate abstractions.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 8: Creating Functions with Parameters

App Lab

Overview

In this lesson, students practice using and creating functions with parameters. Students learn that writing functions with parameters can generalize solutions to problems even further. Especially in situations where feel like you are about to duplicate some code with only a few changes to some numbers, that is a good time to write a function that accepts parameters. In the second half of the lesson, students make a series of modifications to a program that creates an “Under the Sea” scene by adding parameters to functions to more easily add variation to the scene. Lastly, students are introduced to App Lab’s random number functions to supply random values to function calls so the scene looks a little different every time the program runs.

Purpose

Writing functions with parameters is a simple idea, but it traditionally has some devilish details for new learners of programming. The basic idea is that you often want to write a function that takes some input and performs some action based on that input. For example, the turtle function moveForward is much more useful when you can specify how much to move forward (e.g., moveForward(100)), rather than just a fixed amount every time. It’s very common to encounter situations where as a programmer you realize that you basically need a duplicate of some code you’ve already got, but you just want to change some numbers. That’s a good time to write a function with a parameter; the parameter just acts as a placeholder for some value that you plug in at the time you call the function. Just like it’s considered good practice to give descriptive names to your functions, the same is true for the names of the parameters themselves. For example: `drawSquare(sideLength)` is better than `drawSquare(stuff)` .

Agenda

Warm Up (5 mins)

- **Recall the Purpose of Parameters**

Activity (40 mins)

- **Writing Functions with Parameters: Under the Sea**

Wrap Up (5 min)

- **When do you need a function with a parameter?**

Assessment

Extended Learning

View on Code Studio

Objectives

Students will be able to:

- Write functions with parameters to generalize a solution instead of duplicating code.
- Identify appropriate situations for creating a function with parameters.
- Use random numbers as inputs to function calls for the purpose of testing.
- Add parameters to a function in an existing piece of code to generalize its behavior.

Preparation

- [ ] Review functions in the "Debug Commands" toolbox located in App Lab to assist in debugging code. These functions are to the left of the Debug Console.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Functions with Parameters** - Video (**download**)

Vocabulary

- **Parameter** - An extra piece of information passed to a function to customize it for a specific need

Introduced Code

- randomNumber min/max
- function myFunction(n){ //code }
- Call a function with parameters

Teaching Guide

Warm Up (5 mins)

Recall the Purpose of Parameters

Remarks

In the previous lesson, we learned to use a lot of new turtle commands. Some of these commands accept a parameter, or even many parameters, which allow us to pass values to the function. This allowed us to make much more interesting images by specifying precisely how far the turtle should move or turn, and introduced the ability to choose specific pen sizes and colors.

Parameters are a powerful programming construct.

- Suppose we have a whole group of similar problems, like turning the turtle some amount.
- Without a parameter we would need 360 different functions, one for each number of degrees we wanted to turn!
- Parameters allow us to use a single function as a general solution to a whole group of problems.

This is clearly a useful construct to use in our programs, and in today's lesson we're going to learn how to create functions with parameters for ourselves.

Discussion

We want to get back to programming fairly quickly.

Use the Getting Started Activity to briefly recall the benefits of functions with parameters they have already seen and motivate the desire to create them for themselves.

Students will make their own functions with parameters in this lesson.

Activity (40 mins)

Transition to Code Studio

Writing Functions with Parameters: Under the Sea

The levels in Code Studio for this lesson walk students through a number of small exercises that build up to making a small "under the sea" drawing.

Below is a quick reference guide that shows what students are asked to do in each level.

Code Studio levels

Lesson Overview

Student Overview

Functions with Parameters

2

(click tabs to see student view)

Drawing Shapes With Parameters

3 4 5 6

(click tabs to see student view)

Check Your Understanding: Defining Functions

7

(click tabs to see student view)

U3 whats a comment_2018

Student Overview

Commenting Code

Teacher Overview | Student Overview

Validation note: no code is checked on this level. There is no automatic way to check whether a student has added a particular comment into the code.

View on Code Studio

Under the Sea Drawing with Parameters

10 | 11 | 12 | 13 | 14 | 15 *(click tabs to see student view)*

Check Your Understanding: Using Random Numbers

16 *(click tabs to see student view)*

Challenge: Continue Your Drawing With More Functions

17 *(click tabs to see student view)*

AP Practice Response - Managing Complexity

18 *(click tabs to see student view)*

Check Your Understanding

19 | 20 *(click tabs to see student view)*

Wrap Up (5 min)

When do you need a function with a parameter?

Reflection: Students should briefly journal and share responses to the following question:

Prompt: "Develop a rule for deciding when to create a function with a parameter rather than a normal function. Below your rule write a couple sentences justifying your rule."

Share Responses:

Once students have written their responses, have them share with a neighbor and ask a few students to share their rules with the class. Focus in particular on the reasons for creating a function with a parameter, as students will likely hit on the two points above. For example:

Discussion

Having used parameters to generalize the behavior of many functions, students should attempt to formalize their understanding of what parameters are and why they are beneficial. Important points to draw out are:

- Parameters allow the creation of a **generalized solution to a whole class of problems**, rather than individually solving many specific problems.
- Parameters remove the need to create repetitive functions, making code easier to write, read, and change.

- “Whenever you have two functions that basically do the same thing make a function with a parameter instead, to remove the need to duplicate code.”
- “If you need a function to do the same thing a lot of different ways, then use a parameter so one function can do it right away.”

If these ideas don’t come out, then you may make these points yourself and instead ask students where they saw these principles in today’s activities. Point out that every time they added a parameter to a function, they generalized the behavior

of the function without having to rewrite the code.

Assessment

1. Multiple Choice: Which of the following are true?
   - Functions with parameters can be used to prevent the creation of duplicated code.
   - Parameters can only be used once within the body of a function.
   - Parameters generalize the solution of a specific problem.
   - Parameters need not be provided to a function in any particular order.
2. Free Response: “Abstraction” is often used to indicate cases where we focus on a general case and ignore a specific instance of a problem. Given this meaning of the word, how are functions with parameters an example of abstraction?

Extended Learning

Students can extend their digital scene by creating parameterized functions for other components of the scene or adding additional parameters to their current set of functions.

Standards Alignment

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **5.3** - Programming is facilitated by appropriate abstractions.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 9: Looping and Random Numbers

Turtle Programming | App Lab

Overview

Students learn to use random values and looping to create variation in their drawings and quickly duplicate objects they wish to appear in their digital scenes many times. Students will be presented with a version of the for loop which only enables them to change the number of times the loop runs. This block is essentially a "repeat" block and will be presented that way. Students will also be presented with blocks which enable them to choose a random number within a given range. Together these blocks enable students to create more complex backgrounds for digital scenes by randomly placing simple objects within the background of their scene. Students use these tools to step through the Under the Sea exemplar digital scene.

Purpose

Loops are a relatively straightforward idea in programming - you want a certain chunk of code to run repeatedly - but it takes a little practice to get good at controlling loops and recognizing how and where in your programs to use them. The for loop in JavaScript (and many other programming languages) is designed to be used for both simple and sophisticated programming tasks, thus it has a lot of syntax to it that will be explained in the future. In this lesson, the block-based form of the for loop exposed to students is effectively a simple repeat loop - it only lets them change a number that dictates how many times the loop repeats.

Random numbers are also used more (much more) in this lesson as an effective way to experiment with loops. Creating some randomly-generated output with each iteration of the loop is good visual feedback that the loop is running the way you expect. It also helps you explore the ranges of possible outputs, which tells you more about what your program can and cannot do.

Agenda

Getting Started (2 minutes)

- **Loops versus Functions**

Activity (30 minutes)

- **Improving Under The Sea with Loops**

Wrap Up (10 minutes)

- **When to use loops versus functions**

Assessment

Extended Learning

View on Code Studio

Objectives

Students will be able to:

- Use a loop in a program to simplify the expression of repeated tasks.
- Identify appropriate situations in a program for using a loop.
- Use random values within a loop to repeat code that behaves differently each time it is executed.

Preparation

- [ ] Set up the room for computer use
- [ ] Plan how you want students to watch the video on loops

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teacher

- **Unit 3 Lesson 9** - Forum

For the Students

- **Tutorial - Loops** - Video (**download**)

Vocabulary

- **For Loop** - Loops that have a predetermined beginning, end, and increment (step interval).
- **Loop** - The action of doing something over and over again.

Teaching Guide

Getting Started (2 minutes)

Loops versus Functions

Purpose

Students will have an opportunity to explore the process of creating loops by programming with them in App Lab. Use the Getting Started Activity to motivate the need in some programs to repeat commands many times, and foreshadow the fact that random inputs can be used in repeated commands to rapidly develop a variety of behaviors.

Remarks

As we have developed as programmers, we have focused on the process of breaking down large tasks into smaller pieces and assigning each piece a function.

When we break down a large task, often we will find that some portion of the task needs to be repeated many times. As programmers, we would simply call the same function many times. This might work if we need to call the same function five times, but if that function needs to be run 1,000,000 times, we'll need a better solution.

Today we'll be exploring how a programming construct called a loop solves this problem by allowing us to repeat a set of commands many times.

We'll also practice looping through commands many times with random input, therefore giving us many instances of random output.

Just as we saw yesterday, this will be both useful for testing our code and also for developing more variety in our drawings.

At the end of class we'll ask: "When should you use a loop instead of a function, or vice versa?"

Activity (30 minutes)

Improving Under The Sea with Loops

- Today we'll learn how to use something called a "for loop"
- The for loop is a JavaScript programming construct - it looks and works the same here as it does in "the real world".

Code Studio levels

Lesson Overview

Student Overview

Using Loops

2 *(click tabs to see student view)*

Introduction to Loops

3 4 5 6 7 *(click tabs to see student view)*

Using Loops to Add Detail to Your Drawing

8 9 10 11 12 13 14 *(click tabs to see student view)*

Project: Under the Sea

15 *(click tabs to see student view)*

AP Practice Response - Identify the Abstraction

16

(click tabs to see student view)

Check Your Understanding

17 | 18 | 19 *(click tabs to see student view)*

Wrap Up (10 minutes)

When to use loops versus functions

Having used loops to repeatedly run a smaller portion of a larger program, students should attempt to formalize their understanding of when a loop should be used.

Reflection: Students should briefly journal and share responses to the following question:

- **"Develop a rule for deciding when to use a loop within a program. Perhaps think about when to use a loop versus a function. Try to make connections to Top-Down Design in your response. Below your rule, write a couple sentences justifying your rule."**

Share Responses:

Once students have written their responses, have them share with a neighbor and ask a few students to share their rules with the class. Focus in particular on the reasons for using a loop.

For example:

- "Use a loop when there is something you need to do over and over again and it doesn't make sense to split it up any more."
- "We don't want to manually call functions many times in a row. If you're calling the same function many times in a row, it's time to make a loop."

Discussion

Important points to draw out are related to when to use a function v. when to use a loop. While you can get code to funciton similiarly with a function or a loop the general rule of thumb is:

- Write a **function** when you have a piece of code - a procedure - that you might reuse in other places in your program.
  - A function might also help you encapsulate a solution to a common problem in one place that you can call on repeatedly with different inputs (e.g. drawSquare(size))
  - Writing a function is often more of a design decision. You are trying to encapsulate and abstract details so you think about other parts of a larger program
- Write a **loop** when there is something you need to do over and over again and it doesn't make sense to split it up any more.
  - Writing a loop is more of an algorithmic decision - you actually just need repetitious behavior to solve some problem.
  - For example, we don't want to manually call functions many times in a row. If you're calling the same function many times in a row, it's time to make a loop.

Assessment

Project:

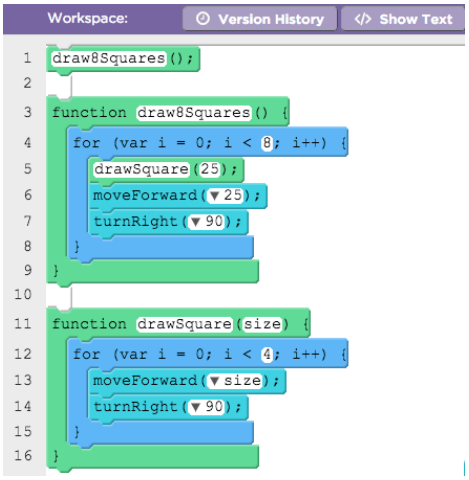
For students' individual renderings of the Under the Sea project, it's recommended that you simply check for completion and that the final version students submit contains all of the required features. In the next lesson, students create their own scene from scratch, and that project has a complete rubric for a more formal assessment.

Questions:

Multiple Choice:

A programmer wants to write a program in which the turtle draws 8 squares in a row while moving forward. The final result should look like the turtle drawing shown at left.

But something is wrong! The incorrect code is shown to the right. The programmer has attempted to write a function to draw a single square. Then he wrote another function that attempts to call that function 8 times. Mentally trace through the code and determine which line of code should be removed to make the program do what it's supposed to.



- A: 1
- B: 5
- C: 6
- D: 7
- E: 14

Free Response:

When breaking a problem down, you often encounter elements that you want to use repeatedly in your code. Sometimes it's appropriate to write a new function; at other times it's appropriate to write a loop. There is no hard-and-fast rule as to which is better, but what do you think? What kinds of circumstances would lead you to writing a function versus using a loop?

Extended Learning

Students can extend their digital scene by manipulating the random values used as inputs to the different components of the scene or adding additional functions for drawing new figures. At the end of the progression, students are given a separate level in which to do so.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **4.1** - Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **5.1** - Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).
- **5.3** - Programming is facilitated by appropriate abstractions.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**

Lesson 10: Practice PT - Design a Digital Scene

Turtle Programming | Project | App Lab

Overview

To conclude their introduction to programming, students will design a program that draws a digital scene of their choosing. Students will be working in groups of 3 or 4 and will begin by identifying a scene they wish to create. They will then use Top-Down Design to identify the high-level functions necessary to create that image. The group will then assign these components to individual members of the group to program. After programming their individual portion, students will combine all of their code to compose the whole scene. The project concludes with reflection questions similar to those students will see on the AP® Performance Tasks.

Note: This is NOT the official AP Performance Task that will be submitted as part of the Advanced Placement exam; it is a practice activity intended to prepare students for some portions of their individual performance at a later time.

AP® is a trademark registered and/or owned by the College Board, which was not involved in the production of, and does not endorse, this curriculum.

Purpose

Abstraction is an important tool in programming, not only because it allows individual programmers to break down complex problems, but because it enables effective forms of collaboration. Once a problem has been broken down into its component parts, teams of programmers (sometimes dozens or more) can attack individual components of that problem in parallel. This style of programming requires clear communication and a shared understanding of the high-level requirements of the software. If implemented carefully, however, it can be an effective strategy for rapidly producing large and complex pieces of software.

Agenda

Getting Started

Activity

- **Review the Project**
- **Complete Group Project Planning Guide**
- **Start Programming**
- **Written Reflection and Program Code**
- **Project Submission**

Wrap Up

View on Code Studio

Objectives

Students will be able to:

- Write programs that address one component of a larger programming problem and integrate with other similarly designed programs.
- Collaborate to break down a complex programming problem into its component parts.
- Use code written by other programmers to complete a larger programming task.
- Write responses to AP-style prompts

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Students

- **Code Print** - Create PT Code Formatting
- **Practice PT - Written Responses** - Student AP Response Template
- **Design a Digital Scene** - Project and Programming Rubric **Make a Copy**
- **Design a Digital Scene - AP Rubric 2019** - AP Rubric **Make a Copy**
- **Design a Digital Scene - Project Guide 2019** - Project Guide **Make a Copy**

Vocabulary

- **Abstraction** - a simplified representation of something more complex. Abstractions allow you to hide details to help you manage complexity, focus on relevant concepts, and reason about problems at a higher level.

Teaching Guide

Getting Started

Goal: Review the programming constructs covered thus far, recall how they can be used within Top-Down Design to break down problems, and frame the coming project as a further exploration of the concept of abstraction.

Activity

Design a Digital Scene

Review the Project

Distribute:

Give each student a copy of the following (digital copies available on code studio):

- **Design a Digital Scene - Project Guide 2019 - Project Guide**
- **Design a Digital Scene - AP Rubric 2019 - AP Rubric**
- **Design a Digital Scene - Project and Programming Rubric**

Read Requirements: Read through the guidelines of the project together and address any high-level questions about the aims of the project. Students will have a chance to review the requirements once placed in groups.

Emphasize and call out connections to the AP Create PT: Students will be going through many of the same processes for this project as they will for the AP Create PT. The things we are not doing here are:

- student is not coming up with the idea for the project - we are framing it as **Design a Digital Scene**
- we are not making a video of code running
- we are ignoring the writing prompts about algorithms

We'll address these things later in Unit 5.

Place Students in Groups: Groups will ideally be 3 or 4 students for this project. You may place students in groups yourself or allow them to select their partners.

Suggested Project

A proposed schedule of the steps of this project is included below, as well as more thorough explanations of how to conduct the various stages.

Day 1

- Review the project guidelines and the rubric.
- Assign students to groups to follow the Group Project Planning guide.
- Groups complete the Project Description document.
- Groups break target scene into high-level functions, define their behavior and complete the Project Component Table.
- Students begin programming individual components.

Day 2

- Students continue to work on programming their individual functions.
- Groups begin to recombine their functions and students begin work on their digital scenes.

Day 3

- Students finalize their digital scenes.
- Students complete their reflection questions and submit their projects.

Complete Group Project Planning Guide

Groups should use the Activity Guide's **Group Project Planning Guide** to collaboratively develop their scene description and select a target image. They will then fill in the Project Component Table with the individual components of the scene, as well as their associated function names and descriptions. Finally, they will assign each function to one of the group members.

Start Programming

How to use the Group Project Planning Guide

We recommend using this guide on the first day of the project to help groups get organized and focus on the project requirements. It is not a document we recommend assessing or evaluating directly, beyond basic completion. See the **project submission** section below for more details.

Code Studio levels

Lesson Overview

Teacher Overview | Student Overview

AP-style student submission

View on Code Studio to access answer key(s)

Exemplar programming projects

These sample App Lab projects contain inline comments explaining how they might be scored against the Project Programming Rubic.

View on Code Studio to access answer key(s)

Please note there is small multi-chose assessment in the next lesson, the "Unit 3 Chapter 1 Assessment." You might use it immediately after, or as part of, evaluating this project.

Q: Where do I find the assessment that used to be linked in this space?

A: It has been moved into the online version you see for the "Unit 3: Chapter 1 Assessment" in the lesson that follows this one.

Design Your Components

2 *(click tabs to see student view)*

How to Share Code with Teammates

3 *(click tabs to see student view)*

Challenge: Design Your Digital Scene

4 *(click tabs to see student view)*

Written Reflection and Program Code

Reflection Questions: Students can complete the reflection questions using the **Practice PT - Written Responses - Student AP Response Template** or simply write them into a plain text document.

Program Code: Students should simply submit their project code through Code Studio by clicking the Submit button where they composed their project.

Project Submission

It is up to you to define and provide instructions for how you want to collect student projects. The things you can collect are:

1. Student written responses - you can collect a digital doc (e.g. Google doc, Word doc), an AP-style PDF sent to you by some means (digital drive, email, etc.) or even a printed hardcopy. Use the **Design a Digital Scene - AP Rubric 2019 - AP Rubric** to score it. An exemplar is available in the teacher-only section on Code Studio.

2. Student code - You should always request they submit the project through Code Studio so it gets locked in. You may **additionally** want to have students mimic the AP requirement of submitting a PDF of their code with the abstraction they developed highlighted with a rectangle drawn around it. We recommend using **CodePrint** to do this. You can collect this as a PDF or ask students to print out a hardcopy.

Teaching Tip

Reflection and Code Submissions: Make a determination of how best students can submit their reflection responses and Code.

If you want to mimic the actual performance task then:

1. Have students write their responses in a document and save as a PDF
2. Have students copy their code into**CodePrint**, draw a rectangle around the abstraction they are writing about, and then print/save that as a PDF

However, if you want to save those extra steps you can have students simply submit a digital document and submit their App Lab project through the Code.org site. You might ask students to identify with a comment in the code the abstraction they wrote about for their written responses.

If you would like to assess the student code for the programming concepts and skills developed in the unit use the **Design a Digital Scene - Project and Programming Rubric** . Exemplars are available in the teacher-only section on code studio.

3. Group planning guide - As mentioned above we do not recommend evaluating or assessing the group planning guide beyond basic completion. If you plan to use the **Design a Digital Scene - Project and Programming Rubric** the planning guide may be a useful artifact or aid for assessing the collaboration aspects of the project.

Wrap Up

Presentation (Optional): If time allows, students may wish to have an opportunity to share their digital scenes with one another. One option is to print out their digital scenes, but consider other options like creating a “Digital Art Gallery” by posting all links to a shared document.

Extended Learning

Incorporate Another Group’s Functions: Students who finish early or wish to have an added challenge can attempt to incorporate the functions of another group into their digital scenes.

Standards Alignment

CSTA K-12 Computer Science Standards (2011)

- **CL** - Collaboration
- **CPP** - Computing Practice & Programming
- **CT** - Computational Thinking

Computer Science Principles

- **2.2** - Multiple levels of abstraction are used to write programs or create other computational artifacts
- **4.1** - Algorithms are precise sequences of instructions for processes that can be executed by a computer and are implemented using programming languages.
- **5.1** - Programs can be developed for creative expression, to satisfy personal curiosity, to create new knowledge, or to solve problems (to help people, organizations, or society).

- **5.3** - Programming is facilitated by appropriate abstractions.
- **5.4** - Programs are developed, maintained, and used by people for different purposes.

CSTA K-12 Computer Science Standards (2017)

- **AP** - Algorithms & Programming
- **IC** - Impacts of Computing

This curriculum is available under a
Creative Commons License (CC BY-NC-SA 4.0).

If you are interested in licensing Code.org materials for commercial purposes,**contact us.**