U

UNPLUGGED

Video Games and Coordinate Planes

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Students discuss the components of their favorite video games and discover that they can be reduced to a series of coordinates. They then explore coordinates in Cartesian space, identifying the coordinates for the characters in a game at various points in time. Once they are comfortable with coordinates, they brainstorm their own games and create sample coordinate lists for different points in time in their own game.

LESSON OBJECTIVES

Students will:

- Create a data model that describes a simple video game.
- Describe the movements of videogame characters by their change in coordinates.

ANCHOR STANDARD

Common Core Math Standards

- **6.NS.8**: Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Reverse Engineering Table (in the student workbook)
- Videogame Design Template (in the student workbook)

For the Teacher

- Lesson slide deck
- Example Game
- Printed cutouts of the Ninja, Dragon, and Unicorn

GETTING STARTED

1) Vocabulary

This lesson has three new and important words:

- **Apply** - use a given function on some inputs
- **Reverse Engineer** - to extract knowledge or design information from an existing product
- **Sprite** - a graphic character on the screen. Sometimes called a bitmap or an image.

2) Learning a Language

Welcome to Code.org CS in Algebra! In this course you'll be learning a new programming language - a way to tell computers exactly what you want them to do. Just like English, Spanish or French, a programming language has its own vocabulary and grammar that you'll have to learn. Fortunately, the language you'll be using here has a lot in common with the simple math that you already know!

Connect this material with things students already know:

- What makes a language?
- Does anyone speak a second (or third) language? Do you speak a different language than your parents/grandparents?
- Are there languages that share features, such as a common root (Romance, Germanic) or a similar alphabet (Latin, Cyrillic, Arabic, Kanji)?
- Are there languages that are designed for specific purposes or within certain constraints (sign language, Esperanto)?
- Math is a language, just like English, Spanish, or any other language!
    - We use nouns, like "bread", "tomato", "mustard" and "cheese" to describe physical objects. Math has values, like the numbers 1, 2 or 3, to describe quantities.
    - We also use verbs like "toast", "slice", "spread" and "melt" to describe operations on these nouns. Mathematics has functions like addition and subtraction, which are operations performed on numbers.
    - Just as you can "slice piece of bread", a person can also "add four and five".

A mathematical expression is like a sentence: it's an instruction for doing something. The expression 4+5 tells us to add 4 and 5. To evaluate an expression, we follow the instructions in the expression. The expression 4+5 evaluates to 9.

ACTIVITIES:

3) Reverse Engineer a Demo

Let's begin by exploring a simple video game, and then figuring out how it works. Open this link to play the game, and spend a minute or two exploring it. You can use the arrow keys to move the up and down - try to catch the unicorn and avoid the dragon!

This game is made up of characters, each of which has its own behavior. The unicorn moves from the left to the right, while the dragon moves in the opposite direction. The ninja only moves when you hit the arrow keys, and can move up and down. We can figure out how the game works by first understanding how each character works.

Directions:

1) Divide students into groups of 2-4.

2) Provide each student with a copy of the reverse-engineering table.

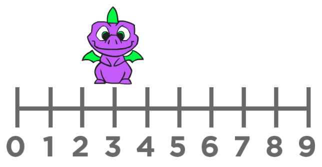
3) As students demo the game, ask them to fill in the "Thing in the game..." column with every object they see in the game.

4) Discuss with the whole group which things they came up with. Characters? Background? Score?

5) Next, for each of the things in the game, fill in the column describing what changes. Size? Movement? Value?

6) Ask students to share back with the whole group. Note how students described changes - how detailed were they? What words did they use to describe movement?

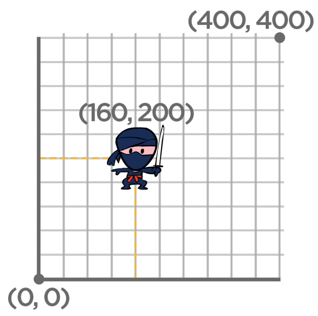
4) Coordinate Planes



Computers use numbers to represent a character's position on screen, using number lines as rulers to measure the distance from the bottom-left corner of the screen. For our video game, we will place the number line so that the screen runs from 0 (on the left) to 400 (on the right). We can take the image of the Dragon, stick it anywhere on the line, and measure the distance back to the left hand edge. Anyone else who knows about our number line will be able to duplicate the exact position of the Dragon, knowing only the number. What is the coordinate of the Dragon on the right hand side of the screen? The center? What coordinate would place the Dragon beyond the left hand edge of the screen?

LESSON TIP

The key point for students here is precision and objectivity. There are many possible correct answers, but students should understand why any solution should be accurate and unambiguous. This requires students to propose solutions that share a common "zero" (the starting point of their number line) and direction (literally, the direction from which a character's position is measured).



By adding a second number line, we can locate a character anywhere on the screen in either dimension. The first line is called the x-axis, which runs from left to right. The second line, which runs up and down, is called the y-axis. A 2-dimensional coordinate consists of both the x- and y-locations on the axes. Suppose we wanted to locate the Ninja's position on the screen. We can find the x-coordinate by dropping a line down from the Ninja and read the position on the number line. The y-coordinate is found by running a line to the y-axis.

A coordinate represents a single point, and an image is (by definition) many points. Some students will ask whether a character's coordinate refers to the center of the image, or one of the corners. In this particular program, the center serves as the coordinate - but other programs may use another location. The important point in discussion with students is that there is flexibility here, as long as the convention is used consistently.

When we write down these coordinates, we always put the x before the y (just like in the alphabet!). Most of the time, you'll see coordinates written like this: (200, 50) meaning that the x-coordinate is 200 and the y-coordinate is 50.

Depending on how a character moves, their position might change only along the x-axis, only along the y-axis, or both. Look back to the table you made. Can the Ninja move up and down in the game? Can he move left and right? So what's changing: his x-coordinate, his y-coordinate, or both? What about the clouds? Do they move up and down? Left and right? Both?

OPTIONAL ACTIVITY: Depending on timing and the background of your students, having one student place a character on a large graph and another student stating the coordinates is excellent practice. Students often need extra practice remembering which coordinate comes first. Coordinates do not have to be exact but they should be in the correct order. Extending this to all four quadrants to include negative numbers is also excellent practice.

Fill in the rest of the reverse-engineering table, identifying what is changing for each of your characters.

WRAP-UP

5) Brainstorming for a Game

Use the game planning template to make your own game. Just like we made a list of everything in the Ninja game, we're going to start with a list of everything in your game. To start, your game will have four things in it:

- A Background, such as a forest, a city, space, etc.
- A Player, who can move when the user hits a key.
- A Target, which flies from the right to the left, and gives the player points for hitting it.
- A Danger, which flies from the right to the left, which the player must avoid.

LESSON TIP

The structure of your students' games will very closely resemble the demo they've just played. Many students will want to reach for the stars and design the next Halo. Remind them that major games like that take massive teams many years to build. Some of the most fun and enduring games are built on very simple mechanics (think Pacman, Tetris, or even Flappy Bird).

2

CS in Algebra | Lesson 2

Evaluation Blocks and Arithmetic Expressions

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Students will begin using Evaluation Blocks to explore the concept of math as a language, and more specifically, a programming language. By composing arithmetic expressions with Evaluation Blocks, students will be able to visualize how expressions follow the order of operations.

LESSON OBJECTIVES

Students will:

- Convert arithmetic expressions to and from code.
- Use Evaluation Blocks to reflect the proper order of operations for an expression.

ANCHOR STANDARD

Common Core Math Standards

- **A.SSE.1**: Interpret expressions that represent a quantity in terms of its context.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Evaluation Blocks Worksheet (in the student workbook)

For the Teacher

- Lesson slide deck

GETTING STARTED

1) Vocabulary

This lesson has five new and important words:

- **Evaluation Block** - a block of code that represents the structure of an expression
- **Evaluate** - perform the computation in an expression, producing an answer
- **Expression** - a computation written in the rules of some language (such as arithmetic, code, or an Evaluation Block)
- **Function** - a mathematical object that takes in some inputs and produces an output
- **Value** - a specific piece of data, like 5 or "hello"

2) Introduction

A mathematical expression is like a sentence: it's an instruction for doing something. The expression 4 + 5 tells us to add 4 and 5. To evaluate an expression, we follow the instructions in the expression. The expression 4 + 5 evaluates to 9.

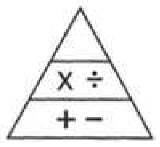
Sometimes, we need multiple expressions to accomplish a task. If you were to write instructions for making a sandwich, it could matter very much which came first: melting the cheese, slicing the bread, spreading the mustard, etc. The order of functions matters in mathematics, too. If someone says "four minus two plus one," they could mean several things:

- Subtract two from four, then add one: **(4 - 2) + 1**
- Add two and one, and subtract the result from four: **4 - (2 + 1)**

Depending on which way you read the expression, you might have very different results! This is a problem, because we often use math to share calculations between people. For example, you and your cell phone company should agree upfront on how much you will pay for sending text messages and making calls. Different results might mean that your bill looks wrong. We avoid problems by agreeing on the order in which to use the different operations in an expression. There are two ways to do this:

1. We can all agree on an order to use
2. We can add detail to expressions that indicate the order

Mathematicians didn't always agree on the order of operations, but now we have a common set of rules for how to evaluate expressions. When evaluating an expression, we begin by applying the operations written at the top of the pyramid (multiplication and division). Only after we have completed all of those operations can we move down to the lower level. If both operations are present (as in 4 - 2 + 1), we read the expression from left to right, applying the operations in the order in which they appear.



Evaluation Blocks provide a visual way to indicate the order of operations in an expression.

All Evaluation Blocks follow three rules:

- Rule 1: Each block must have one function, which is displayed at the top of the block.
- Rule 2: The values for that function are placed below, in order from left to right.
- Rule 3: If a block contains another block as a value, that inner block must be evaluated before the outer block.

Before students get started on the computers, you can have them work through the evaluation blocks worksheet in the student workbook.

ACTIVITY: EVALUATION BLOCKS

3) Online Puzzles

The programming language you are going to learn uses Evaluation Blocks to visually represent mathematical functions. Each block of code is either a Function, or a Value - head to CS in Algebra Stage 2 in Code Studio to get started programming.

3

CS in Algebra | Lesson 3

Strings and Images

Lesson time: 30-60 Minutes

LESSON OVERVIEW

To compute more than just numbers, students will need to learn about two new data types, Strings (any string of alphanumeric characters) and Images. Using these new data types, we'll compose programs that produce and manipulate images.

LESSON OBJECTIVES

Students will:

- Write and evaluate expressions for generating Strings and Images.

ANCHOR STANDARD

Common Core Math Standards

- **A.SSE.1**: Interpret expressions that represent a quantity in terms of its context.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

GETTING STARTED

1) Vocabulary

This lesson has four new and important words:

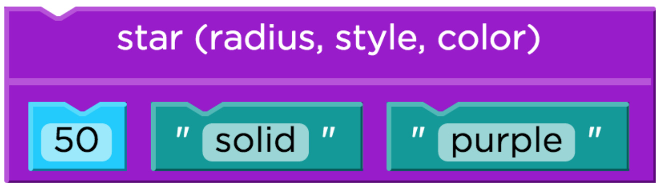
- **String** - any sequence of characters between quotation marks (examples: "hello", "42", "this is a string!")
- **Image** - a type of data for pictures
- **Type** - refers to a general kind of data, like Number, String, Image, or Boolean

2) Introduction

In the previous stage, students only worked with a single type of value - Numbers. In this next stage they will get a chance to write programs with new data types to output text (Strings) and pictures (Images).

Show students the 'star' function, and ask them to discuss the following questions:

- What is the name of this function?
- How many arguments are being given to this function?
- What do you think this function will do?



Students are not expected to know all the answers here - the goal is for them to apply what they know about Evaluation Blocks to a novel expression, and discuss for themselves what they think it might mean. Ask them to justify their answers, and to explain why they think they are correct. Encourage students to look for patterns among these new blocks (such as colors, or quotation marks around the words "solid" and "purple" - what might those patterns mean?

ACTIVITY: STRINGS AND IMAGES

3) Online Puzzles

In this activity you'll use the new data types String and Image to compose art with Blocks of Evaluation - head to CS in Algebra Stage 3 in Code Studio to get started programming.

Derived from

U

UNPLUGGED

Contracts, Domain, and Range

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Contracts provide a way for students to better understand and discuss functions. Through this lesson, students will look at known functions and come up with the contracts that describe those functions.

LESSON OBJECTIVES

Students will:

- Describe a function in terms of its name, domain, and range.
- Create contracts for arithmetic and image-producing functions.

ANCHOR STANDARD

Common Core Math Standards

- **F.IF.1**: Understand that a function from one set (called the domain) to another set (called the range) assigns to each element of the domain exactly one element of the range. If f is a function and x is an element of its domain, then f(x) denotes the output of f corresponding to the input x. The graph of f is the graph of the equation y = f(x).

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

LESSON OBJECTIVES

Students will:

- Describe a function in terms of its name, domain, and range
- Create contracts for arithmetic and image-producing functions

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Contract Log

For the Teacher

- Lesson Slide Deck

GETTING STARTED

1) Vocabulary

This lesson has three new and important words:

- **Contract** - a statement of the name, domain, and range of a function
- **Domain** - the type of data that a function expects
- **Range** - the type of data that a function produces

2) What's in a Function

You've already seen several functions that take in two Numbers, such as +, and -. Other functions like "star", take in a Number and two Strings. Different functions take in different inputs, and we need a way to keep track of the requirements for each function.

- What does the '+' function do?
  - What does it take as input?
  - What does it return as output?
- How about the 'triangle' function?
- What do these different functions have in common?

Let's look at a simple way to describe any function, it's called a "contract"

- What is a Contract?
  - A formal agreement
  - A description of expected behavior
- What do Contracts tell us?
  - What a function should do
  - What inputs a function needs
  - What a function returns

Encourage students to think about contracts in the real world. What purpose do they serve? If a contract is signed, do we expect it to be followed?

Contracts have three distinct parts:

1. Name
2. Domain
3. Range

The Name of a function briefly describes what the function does.

The Domain of a function is the data that the function expects.

The Range of a function is the data that the function produces.

By keeping a list of all the functions in a language, and their Domains, programmers can easily look up how each function is used. However, it’s also important to keep track of what each function produces! For example, a program wouldn’t use "star" if they were trying to produce a Number, because star only produces Images.

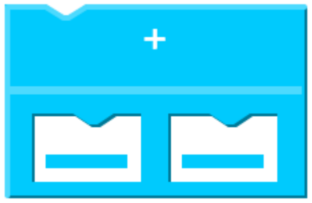
Domains and Ranges help programmers write better code, by preventing silly mistakes and giving themselves hints about what to do next. A programmer who wants to use "star" can look up the Domain and immediately know that the first input has to be a Number (like 100), without having to remember it each time. Instead of writing a single value there, a programmer could write a whole expression, like (25 * 4). We know this code will return an appropriate value (Number) by looking at the Range for *; therefore, the result of * can be used in place of any Number value.

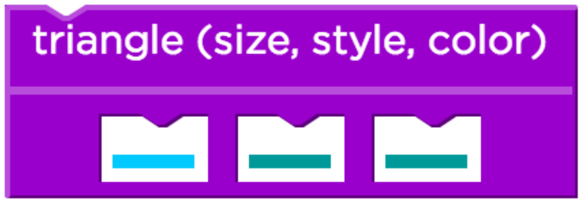
When programmers write down the Domains and Ranges of each function, they write what are called **contracts**, to keep track of what each function needs.

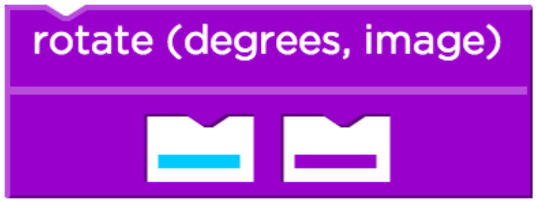
ACTIVITIES:

3) Reading Contracts

Let's look at a few example contracts - for each contract we'll identify the Name, Domain, and Range

- +: Number Number -> Number
- 

- triangle: Number String String -> Image
- 

- rotate: Number Image -> Image
- 

4) Writing Contracts

Let's see if we can come up with contracts for some of the functions you've already seen. You'll want to make sure that you've got your contract log, as this is where you'll keep a running document of all contracts you write - both for existing functions and ones of your own creation.

- We'll start with contracts for simple arithmetic functions
- +, -, *, /

Those were pretty easy as arithmetic functions only deal in Numbers. When it comes to writing functions that deal with multiple data types, looking at the Evaluation Block can give us some helpful clues.

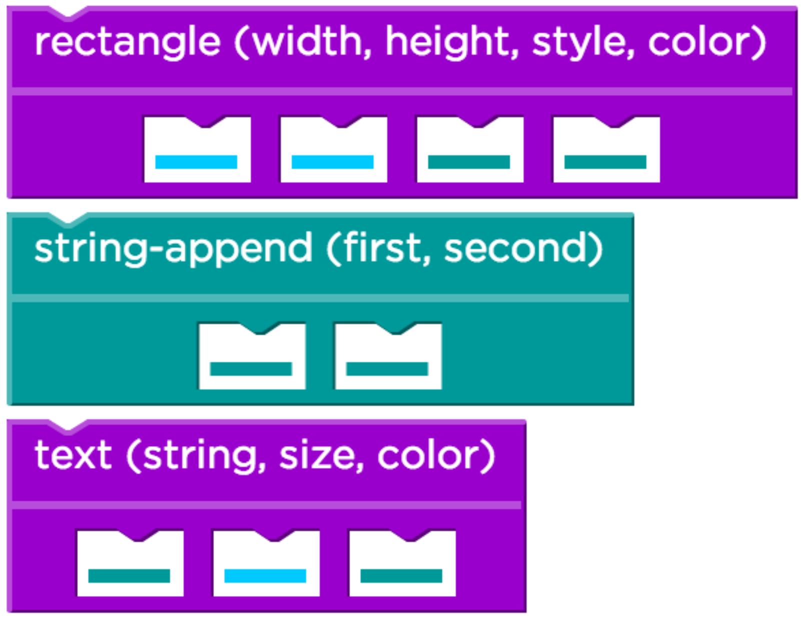
- The Name of each function is at the top
- There will be a slot for each Domain element
- The color of each slot tells you Domain type
- The color of the whole block tells you Range
- Color codes: Number String Image

LESSON TIP

Common mistakes when students first write down contracts include: writing values (such as "red") instead of types (such as "String") and forgetting arguments. Read your students' contracts carefully, as they often indicate misconceptions that will persist and affect them later on.

Display each of the following Evaluation Blocks and ask students:

- What is the Name of this function?
- What is the Domain of this function?
- What is the Range of this function?
- Add this function's contract to your reference


rectangle (width, height, style, color)
string-append (first, second)
text (string, size, color)

WRAP-UP

5) Keep up your Contracts

As you continue programming, make sure that you document a contract for every new function you encounter or write. In the next unit, you'll learn how to create your own functions to save work in writing expressions (this will turn out to be an essential part of writing a game). You'll also start customizing your game with images for the elements in your game design.

Derived from BOOTSTRAP www.bootstrapworld.org

5

CS in Algebra | Lesson 5

Writing Contracts

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Students will work their way through a number of new functions, first using each to solve a problem, and then writing a contract which describes it.

LESSON OBJECTIVES

Students will:

- Decompose existing functions.
- Write contracts that describe functions.
- Experiment with basic geometric transformations.

ANCHOR STANDARD

Common Core Math Standards

- **8.G.1**: Verify experimentally the properties of rotations, reflections, and translations:

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

GETTING STARTED

1) Vocabulary

This lesson has three new and important words:

- **Rotate** - to turn a shape about a point.
- **Scale** - to increase the dimensions of a shape by the same factor in all directions. Also known as dilate.
- **Translate** - to move a shape from one location to another. The **offset** function performed this transformation.

2) Introduction

Review with students the purpose of a Contract:

- Describes three elements of a function
  - Name (what is the function called)
  - Domain (what inputs does it take)
  - Range (what does it output)
- As a class, describe the Contracts for some basic mathematical operators
  - Addition (name +, domain Number Number, range Number)
  - Subtraction (name -, domain Number Number, range Number)
  - Multiplication (name *, domain Number Number, range Number)
  - Power of two (name sqr, domain Number, range Number)

ACTIVITY: WRITING CONTRACTS

3) Online Puzzles

In this stage you'll be looking at some functions, some of which you've seen before and some which are brand new. For each function you'll first get a chance to use the function, and then you'll write a Contract for it. Make sure to document any new Contracts on your Contract Log. Head to CS in Algebra stage 5 in Code Studio to get started programming.

Derived from

6

CS in Algebra | Lesson 6

Defining Variables and Substitution

Lesson time: 30-60 Minutes

LESSON OVERVIEW

In this activity, students will learn to define variables that can be used to reference values and expressions. Once defined, their variables can be used repeatedly throughout a program as substitutes for the original values or expressions.

LESSON OBJECTIVES

Students will:

- Define variables by giving them a name and assigning them a value or expression.
- Use variables within Evaluation Blocks.
- Describe a situation where using variables as substitutions for values or expressions is more efficient.

ANCHOR STANDARD

Common Core Math Standards

- **6.EE.4**: Identify when two expressions are equivalent (i.e., when the two expressions name the same number regardless of which value is substituted into them). For example, the expressions y + y + y and 3y are equivalent because they name the same number regardless of which number y stands for.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

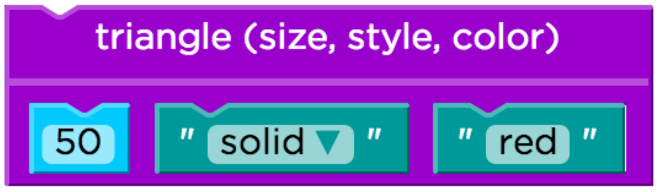
GETTING STARTED

1) Vocabulary

This lesson has two new and important words:

- **Define** - associate a descriptive name with a value
- **Variable** - a container for a value or expression that can be used repeatedly throughout a program

2) Introduction



Suppose we want to make an image with fifty identical, solid red triangles. To do so you'd have to create this Evaluation Block fifty times!

Even worse, if you decided you wanted fifty blue triangles instead, you'd have to go through and change each and every block. There must be a better way!

We can store that red triangle Evaluation Block in a Variable, let's call it "red-triangle." That name "red-triangle" now becomes a shortcut for the blocks inside the variable, and we can use that shortcut over and over in our program. If we decide that we want that red triangle to be 100 pixels instead of 50, we only need to change it in the variable definition.

LESSON TIP

*If students have used variables in other programming languages, it's essential to note that in functional programming, as in math, variables are considered **immutable** - meaning the value can't be changed during the execution of a program. Think about it this way: saying x = 50, and then x = x + 1 might make sense in Javascript, but it's impossible in Algebra.*

ACTIVITY: DEFINING VARIABLES AND SUBSTITUTION

3) Online Puzzles

In this stage you'll use variables to reference a variety of values and expressions. Head to CS in Algebra stage 6 in Code Studio to get started programming.

Derived from BOOTSTRAP www.bootstrapworld.org

7

CS in Algebra | Lesson 7

The Big Game - Variables

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Students get their first look at the inside of their own video games. They will start development by substituting in new Images, Strings, and Numbers for existing variables.

LESSON OBJECTIVES

Students will:

- Substitute new values into existing variables of an existing program and describe the effects.
- Examine the structure of an existing program.

ANCHOR STANDARD

Common Core Math Standards

- **6.EE.4**: Identify when two expressions are equivalent (i.e., when the two expressions name the same number regardless of which value is substituted into them). For example, the expressions y + y + y and 3y are equivalent because they name the same number regardless of which number y stands for.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

GETTING STARTED

1) Vocabulary

This lesson has three new and important words:

- **Troubleshooting** - when a program generates an unexpected result, a programmer must examine the code to determine the source of the unexpected results (usually an unanticipated input or incorrect handling of an expected input). Sometimes called **debugging**.
- **Mod** - Short for modification. Games in the real world are often a mod of another game. Othello (or Reversi) is

usually considered a mod of the ancient game of “Go”. A mod of a program is one that has been altered to do something slightly different than its original purpose.

- **Stub** - A function whose domain and range have been designated, but the process to transform the domain into the range has not yet been defined.

2) Teaching Notes on the Big Game

The students will create a mod of an existing game. As they make changes to the game, it is possible that they will add code that will either “break” the program (cause nothing to happen) or cause an unexpected wutcome. If either of these conditions exist, they will need to troubleshoot or debug the code to determine how to get it working in the proper way. **If things go terribly awry and finding a problem is too frustrating, use the Clear Puzzle button in the upper right corner of the workspace.** This button will clear your game back to its initial state, so it should only be used as a last resort.

This exercise is a simplified version of a very common real world programming task. Programmers often create mods of programs about which they know very little. They slowly unravel which pieces require further understanding in order to make the mod work the way they want, while leaving other parts of the program completely unexplored.

Many programs and functions are customizable through their arguments (which can be variables or values). When a function is called, its arguments are passed in as variables into the function. In other cases, variables that someone might want to change (sometimes called constants) are often at the top of a piece of code. Having access to the code allows the programmer to change the way the program behaves by setting these variables to different values.

In this lesson, we are creating the mod by changing the variables inside the code. The student has access to the game code and is changing the initial value of the Title, Subtitle, Player, Danger, and Target. As a reminder, the **ultimate** goal of this game will be to manipulate the player through pressing keys, to avoid the danger, and to make contact with the target. The **current** lesson has no motion or interactivity. It only changes the look of the game. The motion and interactivity function stubs, such as “update-target” and “danger?”, will be completed in later lessons.

The blocks menu displays a few new items (Boolean, Cond, and Functions) which will be examined in more detail in future lessons. The students should be encouraged to explore each of the sub-menus. However **the only navigation required for this level is editing the five color blocks at the top of the function:** Title, subtitle, bg (background), player, target, and danger. The difference between the color and black blocks will also be explained in a future lesson.

ACTIVITY: THE BIG GAME - VARIABLES

3) Online Puzzles

In this stage you'll define and modify variables to changes how some games function. Head to CS in Algebra stage 7 in Code Studio to get started programming.

Derived from

BOOTSTRAP
www.bootstrapworld.org

8

CS in Algebra | Lesson 8

Composite Functions

Lesson time: 30-60 Minutes

LESSON OVERVIEW

In the past lessons students have defined variables which will allow them to easily write expressions that refer to the same value repeatedly. In this stage, they will write simple functions that, like variables, allow students to abstract out repetitious elements of their programs.

LESSON OBJECTIVES

Students will:

- Analyze and use existing functions.
- Modify existing functions.
- Create new functions.
- Create similar shapes by changing size parameters on functions.

ANCHOR STANDARD

Common Core Math Standards

- **8.F.1**: Understand that a function is a rule that assigns to each input exactly one output. The graph of a function is the set of ordered pairs consisting of an input and the corresponding output.1

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

GETTING STARTED

1) Vocabulary

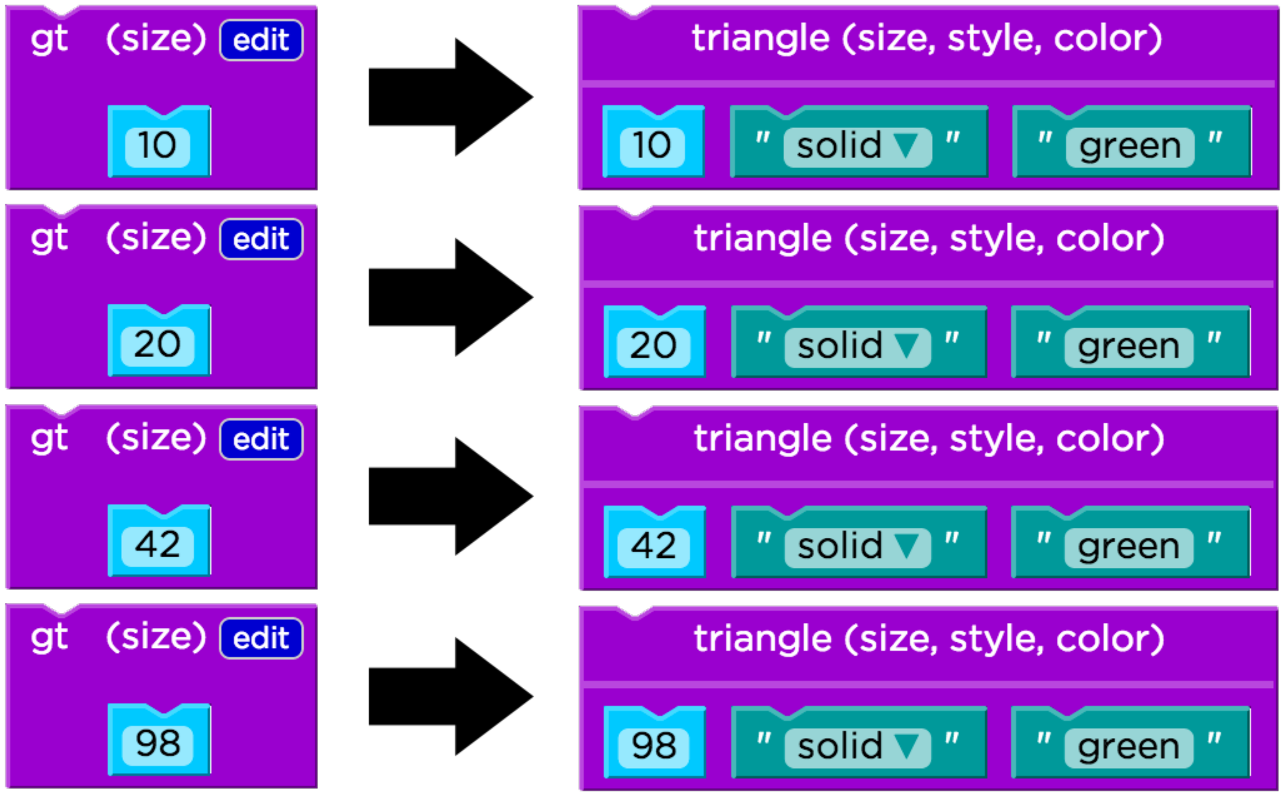
This lesson has one new and important word:

- **Parameter** - A value or expression belonging to the domain.

2) Introduction

Defining a reusable *value* is helpful when a program has lots of identical expressions. Sometimes, however, a program has expressions that aren't identical, but are just *very similar*. A program that has fifty solid, green triangles can be simplified by defining a single value, *as long as they are all the same size*. But what if a program has fifty solid, green triangles of different sizes?

Think about the Image functions you have already used, like star and circle. They take inputs and produce images. Similarly, we might want a green-triangle function that takes the size as an input and produces a green triangle. The programming language doesn't provide this function, but it does let you define your own functions. We want to define our own function (let's call it gt, for green triangle) that takes in a Number as the size parameter and produces a solid green triangle of whatever size we want. For example:



and so on...

ACTIVITY: COMPOSITE FUNCTIONS

2) Online Puzzles

In this stage you'll define simple functions. Head to CS in Algebra stage 8 in Code Studio to get started programming.

Derived from

www.bootstrapworld.org

U

UNPLUGGED

The Design Recipe

Lesson time: 30-60 Minutes

LESSON OVERVIEW

In the last stage, students wrote some very simple functions - but more sophisticated functions demand a more thoughtful approach. The Design Recipe is a structured approach to writing functions that includes writing test cases to ensure that the function works as expected. Once students have mastered the Design Recipe process, they can apply it to any word problem they encounter.

LESSON OBJECTIVES

Students will:

- Use the Design Recipe to identify dependent variables, independent variables, and constants.

ANCHOR STANDARD

Common Core Math Standards

- **F.BF.1**: Write a function that describes a relationship between two quantities.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Fast Functions Sheet
- Blank Design Recipe Form

For the Teacher

- Lesson Slide Deck

GETTING STARTED

1) Vocabulary

This lesson has five new and important words:

- **Design Recipe** - a sequence of steps to document, test, and write functions.
- **Purpose Statement** - a brief description of what the function does.
- **Independent Variable** - The value that the experimenter controls. The input.
- **Dependent Variable** - The value that changes based on the independent variable. The output.
- **Constant** - A fixed number in a relationship.

2) What is the Design Recipe

The Design Recipe is a roadmap for defining functions, which programmers use to make sure the code they write does what they want it to do. Each step builds on the last, so any mistakes can be caught early in the process. This roadmap has a series of steps:

1. Write a Contract that describes the word problem.
2. Write Examples based on the contract.
3. Define a function that matches the examples.

Let's start out by applying the Design Recipe together to the following problem:

Define a function ’purple-star’, that takes in the size of the star and produces an outlined, purple star of the given size.

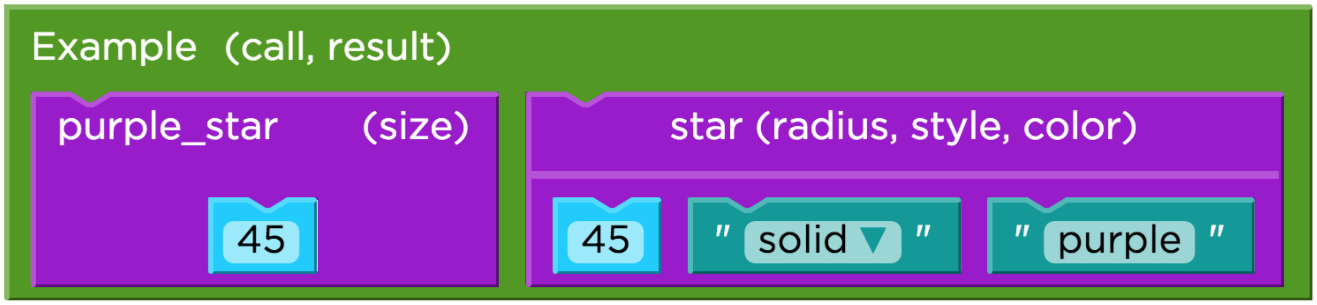
Step 1 - The Contract

purple-star: Number -> Image

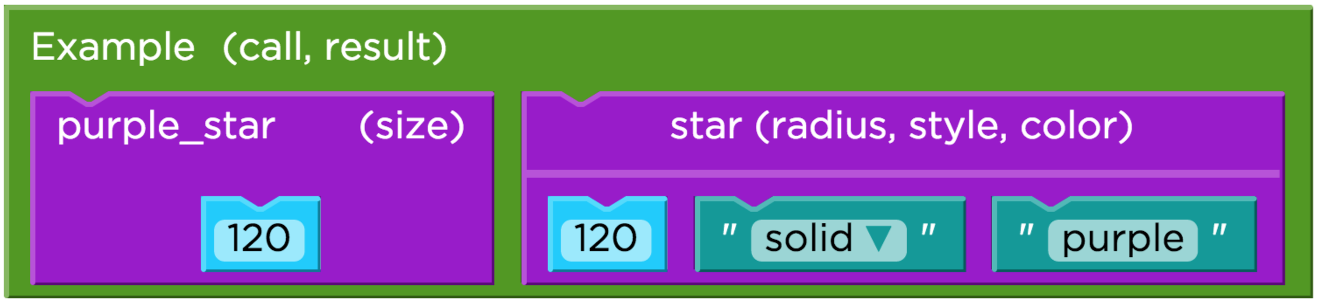
Be sure to include a good Name for each function, and remember that the Domain and Range can only include types like Numbers, Images, Strings, etc.

A Contract is the foundation for a function, which gives programmers just enough information to use it: the name of the function, the type (or types) of data it expects and the type of data it returns.

Step 2 - Examples



```
Example (call, result)
purple_star (size)          star (radius, style, color)
  45                          45   " solid ▼ "   " purple "
```



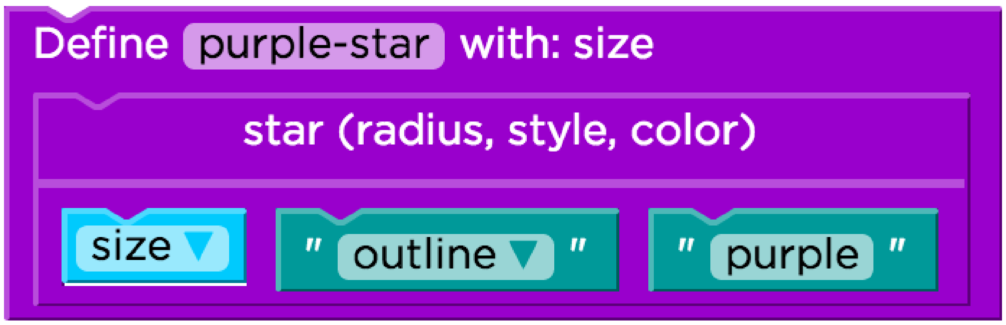
```
Example (call, result)
purple_star (size)          star (radius, style, color)
  120                         120  " solid ▼ "   " purple "
```

- Every Example begins with the name of the function. Where could you find the name of the function?
- Every Example has to include sample inputs. Where could you find out how many inputs this function needs,

and what types they are?
- Every Example has to include an expression for what the function should do when given an input. Where could you look to find out what this function does?

Once you have two or more Examples, it should be easy to identify what has changed between them. In fact, the number of things that change should match the number of things in the function’s Domain: if the Domain has a Number and a String in it, then those two values should be the things that differ between your Examples.

Step 3 - Function Definition



```
Define purple-star with: size
    star (radius, style, color)
    size    " outline "    " purple "
```

By identifying what has changed between these Examples, we can define our actual function.

Challenge students to explain why this function does not need to know the color of the star, or whether or not it is solid. The main idea here is that the function already "knows" these things, so the only thing that is changing is the size of the star.

Remember that the Contract and Purpose Statement can be used to write the Examples, even if a programmer isn’t sure how to begin.

ACTIVITIES:

3) Collaborative Design

- Define a function ’spot’, that takes in a color and produces a solid circle of radius 50, filled in with that color.
- Define a function ’average’, which takes in two numbers and produces their average. (You may need to remind the students that to find the average of two numbers, they should be added together and divided by two.)
- Suppose a company logo is a word drawn in big, red letters, rotated some number of degrees. Define a function ’logo’, that takes in a company name and a rotation, and produces a logo for that company.

Put students into groups of 3 - each member of the group will represent one step of the Design Recipe

1. Contract
2. Examples
3. Function

Each group will work through a set of word problems using the Fast Functions Sheet. We recommend that you pull word problems from your own curriculum so that students can see how the Design Recipe can be used outside of programming. Make sure that each group member stays true to their role and that they work through the steps in the right order. If you don't have problems to use from your curriculum, there are a number of examples available in this lesson's slide deck.

LESSON TIP

Challenge students to explain their Examples (their function name, the number of inputs, their types and the type of the returned value). Make sure that the two Examples for each function have different input values! For each of these questions, students must be able to point to the specific part of their Contract as the justification for their Example.

Make sure students have chosen good variable names for their function definitions, and ask students to justify every part of the function body. The only acceptable answers should be "I copied this because it’s the same in both Examples", or "I used *a ariable name because it differs bet eenxamples."*

Once students have worked through the Fast Functions, you can have them use the full Blank Design Recipe Form to work through an word problems that they encounter in the future.

Derived from

BOOTSTRAP
www.bootstrapworld.org

10

CS in Algebra | Lesson 10

Rocket Height

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using the Design Recipe, students will work through a series of word problems about calculating the height of a rocket after a given number of seconds from launch. The functions they write will be used to animate the rocket launch.

LESSON OBJECTIVES

Students will:

- Design functions to solve word problems.
- Use the Design Recipe to write contracts, test cases, and function definitions.

ANCHOR STANDARD

Common Core Math Standards

- **F.LE.1**: Distinguish between situations that can be modeled with linear functions and with exponential functions.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

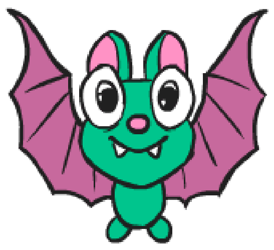
- Rocket-Height Design Recipe (in the student workbook)

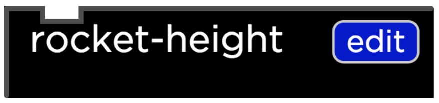
GETTING STARTED

1) Introduction

Functions are a key part of animation in computer programs. A function that draws a static picture of a bat, for example, can place the bat at a different location based on the input. When that input changes slightly based on time or user-interaction, the bat will appear to move. This is similar to the way that flip-book animations work, in which each page draws a static image that has changed by a small amount. When the pages are displayed quickly, the images appear to change smoothly.

Putting these images together, we arrive at an animation of the bat turning around.





In the online puzzles, students will find a black block for each function they create, in addition to the colored blocks they are used to. The black function box, which has no parameter inputs, represents the function as a Type of data. This allows you to pass your function into the 'start' function, where it can be used to control the rocket animation.

Another curiosity with this program is that the rocket-height function will be executed multiple times. The periodic execution creates the flip-book effect. As each second passes, the rocket-height function is executed again, the new location is calculated, and the rocket is re-drawn in its new location. This drawing and re-drawing in different locations gives the appearance of motion.

LESSON TIP

After creating simple linear movement, students will be asked to write functions to animate simple acceleration. Students will be given an input/output table from which to write their new function. You may want to work through these problems as a whole class, so that students can see how you might analyze an input/output table in order understand the relationship between input and output values.

ACTIVITY: ROCKET HEIGHT

2) Online Puzzles

In this stage you'll write functions that manipulate images to create animations. Head to CS in Algebra stage 10 in Code Studio to get started programming.

EXTENSION ACTIVITIES

3) Non-linear Animation

The final puzzle of this stage is a Free Play puzzle that will allow you amd your students to experiment with other variations on the rocket-height formula. One activity that students find particularly interesting (and often challenging) is to write functions that produce non-linear acceleration. If your students are familiar with quadratics then you can call this out as such, but even younger students who haven't yet seen quadratics can enjoy this extension challenge.

Place the following input/output tables on the board and see if students can come up with functions that will produce the appropriate animation.

Challenge 1

Input	Output
1	10
2	40
3	90
4	160

Challenge 2

Input	Output
1	15
2	45
3	95
4	165

Once students have figured out the provided Input Output tables, encourage them to come up with non-linear animation functions of their own.

CODE

Derived from BOOTSTRAP www.bootstrapworld.org

11

CS in Algebra | Lesson 11

Solving Word Problems with the Design Recipe

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Students will continue to practice the Design Recipe with a series of word problems.

LESSON OBJECTIVES

Students will:

- Design functions to solve word problems.
- Continue to practice writing contracts with more complex scenarios.

ANCHOR STANDARD

Common Core Math Standards

- **F.BF.1**: Write a function that describes a relationship between two quantities.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

GETTING STARTED

1) Introduction

The students will do lots of dragging and dropping as they fill in the missing pieces of different parts of various contracts. It should be noted that the examples must be filled in completely. The error message when the example is incomplete is "You have a block with an unfilled input."

ACTIVITY: SOLVING WORD PROBLEMS WITH THE DESIGN RECIPE

2) Online Puzzles

In this stage you'll use the Design Recipe to create functions that solve word problems. Head to CS in Algebra stage 11 in Code Studio to get started programming.

Derived from

12

CS in Algebra | Lesson 12

The Big Game - Animation

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Returning to the Big Game we started in stage 7, students will use the Design Recipe to develop functions that animate the Target and Danger sprites in their games.

LESSON OBJECTIVES

Students will:

- Design functions to solve word problems.
- Use the Design Recipe to write contracts, test cases, and function definitions.

ANCHOR STANDARD

Common Core Math Standards

- **F.LE.2**: Construct linear and exponential functions, including arithmetic and geometric sequences, given a graph, a description of a relationship, or two input-output pairs (include reading these from a table).

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Update-target Design Recipe (in the student workbook)
- Update-danger Design Recipe (in the student workbook)

GETTING STARTED

1) Introduction

Let's get back into that Big Game that we started in stage 7.

The primary goal here is to get the *target* (starting in the upper left) to travel from left to right and the *danger* (starting in the lower right) to travel from right to left. This is accomplished in the update-target and update-danger blocks by changing the output of the function from its current default value of an unchanging x to some value relative to x.

Similar to the rocket-height puzzle, the update-target and update-danger functions are executed in order about every 10th of a second, to create the flip-book effect of movement. Each time these updates are executed, the functions take the CURRENT x coordinate as input and then return a new x coordinate such that the image's position changes. For each new execution of the update, the x coordinate set by the previous execution becomes the starting point.

One new thing the students should notice is that their modifications from stage 7 should still be in place. The Big Game will save a file for each student, and each level that they work on will benefit from the the changes made in previous levels. This means that it is very important that every student gets each Big Game level working correctly before moving on to the next stage.

It should also be noted that if a student returns to a previous level, or even a previous stage, that the MOST RECENT changes which they made will be the ones that they will see. Backing up to a previous level does NOT restore the previous state of the student's Big Game. Students are always looking at their most recent changes no matter which puzzle they are in.

LESSON TIP

A contract can be quite long and often scrolls off the screen. To make dragging into the Definition area easier, consider collapsing the "1. Contract" and "2. Examples" areas by clicking on the arrow to the left of them.

ACTIVITY: THE BIG GAME - ANIMATION

2) Online Puzzles

Using what you've learned about the Design Recipe you'll be writing functions that add animation to your game. Head to CS in Algebra stage 12 in Code Studio to get started programming. Note that when you click run, the title and subtitle will display for about 5 seconds before the other functions start.

U

UNPLUGGED

Booleans and Logic

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Booleans are the fourth and final data type that students will learn about in this course. In this stage, students will learn about Boolean (true/false) values, and explore how they can be used to evaluate logical questions.

LESSON OBJECTIVES

Students will:

- Evaluate simple Boolean expressions.
- Evaluate complex Boolean expressions.

ANCHOR STANDARD

Common Core Math Standards

- **7.EE.4**: Use variables to represent quantities in a real-world or mathematical problem, and construct simple equations and inequalities to solve problems by reasoning about the quantities.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

Getting Started

1) Vocabulary
2) Booleans - True or False?

Activity: Booleans 20 Questions

3) Boolean 20 Questions

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Teacher

- Lesson Slide Deck

For the Students

- 3 x 5 cards, pens or pencils

GETTING STARTED

1) Vocabulary

This lesson has two new and important words:

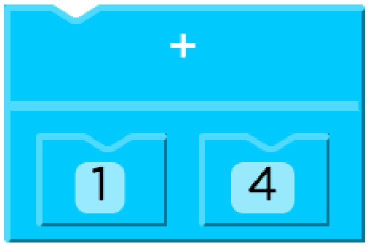
- **Boolean** - a type of data with two values: true and false.
- **Return** - used as a synonym for output of a function.

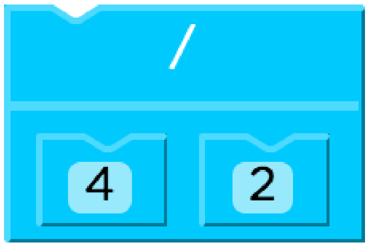
2) Booleans - True or False?

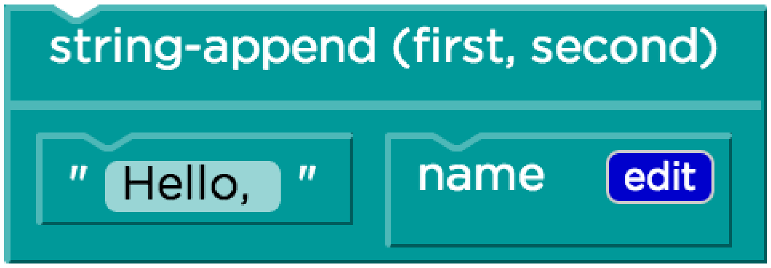
What types of data have we used in our programs so far?

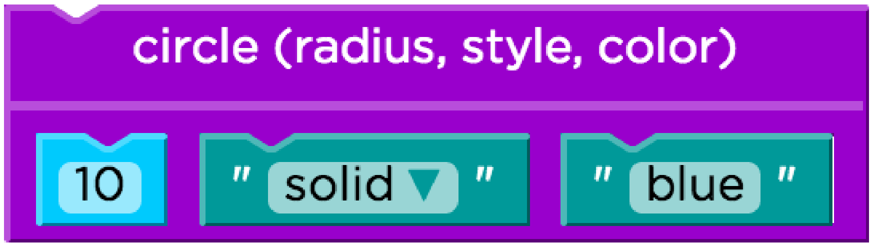
- Can you think of Number values?
- String values? Image values?
- What are some expressions that evaluate to a Number?
- How about the other datatypes?

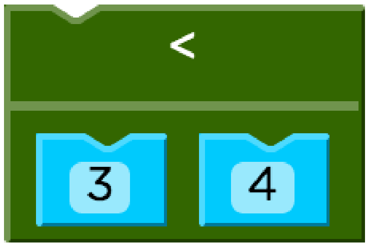
What would each of the following expressions evaluate to?











The last expression, **(3 < 4)**, uses a new function that compares Numbers, returning **true** if 3 is less than 4. What do you think it would return if the numbers were swapped?

The function < tests if one number is less than another. Can you think of some other tests?

Functions like **<**, **>** and **=** all consume two Numbers as their Domain, and produce a special value called a Boolean as their Range. Booleans are answers to a yes-or-no question, and Boolean functions are used to perform tests. In a videogame, you might test if a player has walked into a wall, or if their health is equal to zero. A machine in a doctor's office might use Booleans to test if a patient's heart rate is above or below a certain level.

Boolean values can only be true or false.

ACTIVITIES:

3) Boolean 20 Questions

Give each student a card and have them answer the following questions on it (feel free to add some of your own)

1. What is your hair color?
2. Do you wear glasses or contacts?
3. What is your favorite number?
4. What is your favorite color?
5. What month were you born?
6. Do you have any siblings?
7. What is the last digit of your phone number?
8. What is something about you that people here don't know and can't tell by looking at you?

Then collect the cards and shuffle them. To play the game, follow these steps:

- Select a card
- Say: I'm going to read the answer to #8 but if it is you, don't say anything.
- Say: Now everyone stand up and we are going to ask some questions with Boolean answers to help determine who this person is.
- Begin the following true/false questions. Preface each one with "If you answer false to the following question, please sit down." The person whose card you are reading should always answer true so you will need to change the example questions below. For this example, the answers were:

1. What is your hair color? - **brown**
2. Do you wear glasses or contacts? - **yes**
3. What is your favorite number? - **13**
4. What is your favorite color? - **blue**
5. What month were you born? - **December**
6. Do you have any siblings? - **yes**
7. What is the last digit of your phone number? - **7**

With that example, you might make the following statements:

- My hair color is brown.
- I wear contacts or glasses. (you only have to answer true to One of these to remain standing)
- My favorite number is greater than 10 and less than 20. (you must answer true to both these.)
- My favorite color is blue or green.
- I was not born in April.
- I have at least one sibling.
- The last digit of my phone number is a prime number.

Because of how numbers 3,4, 5, and 7 were asked it is likely that some people will still be standing. You will need to revisit these and ask them again in a more narrow fashion such as "My favorite color is blue".

Play this several times. Be creative with using *or*s and *and*s. Remind students that the OR means that either part of the statement being true will result in the entire statement being true. In English, an "or" is often an "exclusive or" such as "You can have chicken or fish." In English, you only get to pick one, but with Boolean logic you could have chicken, fish, or both!! For the example person above, "I was born in December OR my favorite number is

13” is true. Note that “I was born in December AND my favorite number is 13” is also true.

Have a student try to act as the quizmaster after several rounds. If a mistake is made by you, a student quizmaster, or the person whose card you are reading, see if you can analyze where the mistake was made or why the question being asked might not have been clear.

How does this activity connect with our game? In our game, we may need to determine: Is a target too far left or too far right? If so, then perhaps some action should occur.

14

CS in Algebra | Lesson 14

Boolean Operators

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using Boolean operators, students will write code that compares values to make logical decisions.

LESSON OBJECTIVES

Students will:

- Use Boolean operators to compare values.
- Apply Boolean logic, such as AND, OR, and NOT, to compose complex Boolean comparisons.

ANCHOR STANDARD

Common Core Math Standards

- **7.EE.4**: Use variables to represent quantities in a real-world or mathematical problem, and construct simple equations and inequalities to solve problems by reasoning about the quantities.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Teacher

- Lesson slide deck

GETTING STARTED

1) Introduction

Creating some sample boolean expressions - both simple and complex - is an excellent warm-up activity before the puzzle stages. Some examples have been included in the slide deck. The slide deck also has extra practice related to expressions that the students will have seen in the puzzles.

ACTIVITY: BOOLEAN OPERATORS

2) Online Puzzles

Head to CS in Algebra stage 14 in Code Studio to get started programming.

Derived from

15

CS in Algebra | Lesson 15

Sam the Bat

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using Boolean operators, students will write code that checks the location of a sprite to make sure it doesn't go off-screen.

LESSON OBJECTIVES

Students will:

- Use Boolean operators to compare values.
- Apply Boolean logic, such as AND, OR, and NOT, to compose complex Boolean comparisons.

ANCHOR STANDARD

Common Core Math Standards

- **6.NS.8**: Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

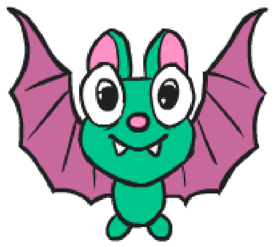
For the Student

- Safe-left? Design Recipe (in the student workbook)
- Safe-right? Design Recipe (in the student workbook)
- Onscreen? Design Recipe (in the student workbook)

GETTING STARTED

1) Introduction

This is Sam the Bat, and his mother tells him that he's free to play in the yard, but he don't set foot (or wing) outside the yard! Sam is safe as long as he is always entirely onscreen. The screen size is 400 pixels by 400 pixels, so how far can Sam go before he starts to leave the screen?



In this stage students write functions that will take in Sam the Bat's next x-coordinate and a return a boolean. That function should return *true* if part of Sam will still be visible, or *false* if he would go too far off-screen. If the function returns *false*, Sam isn't allowed to move.

Students will start by writing functions to check the left and right side of the screen independently, before combining those with a single onscreen? function that prevents Sam from leaving on both the left and right.

For each stage, make sure students try to get Sam to leave through the side they are checking. If Sam makes it all the way off-screen when he shouldn't, they'll get an error, but if he is successfully stopped they'll succeed and move to the next puzzle.

LESSON TIP

It's extremely valuable in this stage to have three students stand, and act out each of these three functions: - Ask each student to tell you their Name, Domain and Range. If they get stuck, remind them that all of this information is written in their Contract! - Practice calling each function, by saying their name and then giving them an x-coordinate. For example, "safe-left? fifty" means that the number 50 is being passed into safe-left?. That student should return "true", since the code currently returns true for all values of x. - Do this for all three functions, and have the class practice calling them with different values as well. Note: the volunteer for onscreen? should first call safe-left?, before replying with the value.

Why not write just one function?

Some students may wonder why they should write separate functions for safe-left? and safe-right? when onscreen? could just check the dimensions of the screen directly. There is more to being a writer than good spelling and grammar. There's more to being an architect or an artist than building a bridge or coloring in a canvas. All of these disciplines involved an element of *design*. Likewise, there is more to being a Programmer than just writing code.

Suppose you just built a car, but it's not working right. What would you do? Ideally, you'd like to test each part of the car (the engine, the transmission, etc) one at a time, to see which one was broken. The same is true for code! If you have a bug, it's much easier to find when every function is simple and easy to test, and the only complex functions are just built out of simpler ones. In this example, you can test your safe-left? and safe-right? functions independently, before stitching them together into onscreen?.

Another reason to define multiple, simple functions is the fact that it lets programmers be lazy. Suppose you have a few characters in a videogame, all of which need to be kept on the screen. Some of them might only need safe-left?, others might only need safe-right?, and only a few might need onscreen?. What happens if the game suddenly needs to run on computers with differently-sized monitors, where the width is 1000 pixels instead of 400? If you have simple and complex functions spread throughout your code, you'll need to change them all. If your complex functions just use the simpler ones, you'd only need to change them in one place!

Badly designed programs can work just fine, but they are hard to read, hard to test, and easy to screw up if things change. As you grow and develop as a programmer, you'll need to think beyond just "making code work". It's not good enough if it just works - as artists, we should care about whether or not code is *well designed*, too. This is what functions allow us to do! Everyone from programmers to mathematicians uses functions to carve up complex problems into simpler pieces, which make it possible to design elegant solutions to difficult problems.

ACTIVITY: SAM THE BAT

2) Online Puzzles

Using Boolean logic, you're going to write functions to help make sure Sam the Bat doesn't leave his mom's yard. Head to CS in Algebra stage 15 in Code Studio to get started programming.

EXTENSION ACTIVITIES

3) Safe up and down

The final puzzle of this stage is a Free Play puzzle that will allow you amd your students to experiment with other ways to keep Sam in his yard. The basic activity only prevents Sam from leaving on the left and right, but what about the top and bottom of the screen?

If you add a second variable to the onscreen? function to take in Sam's y coordinate, then you can check Sam's position on each axis. As students pursue this extension, encourage them to think about how they wrote small component functions to check the left and right. Could you follow a similiar approach to deal with the top and bottom?

Derived from BOOTSTRAP www.bootstrapworld.org

16

CS in Algebra | Lesson 16

The Big Game - Booleans

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using the same logic from the previous lesson, students will write code that checks whether their Target and Danger sprites have left the screen. If their function determines that a sprite is no longer visible on screen, it will be reset to the opposite side.

LESSON OBJECTIVES

Students will:

- Use Boolean operators to compare values.
- Apply Boolean logic, such as AND, OR, and NOT, to compose complex Boolean comparisons.

ANCHOR STANDARD

Common Core Math Standards

- **6.EE.9**: Use variables to represent two quantities in a real-world problem that change in relationship to one another; write an equation to express one quantity, thought of as the dependent variable, in terms of the other quantity, thought of as the independent variable. Analyze the relationship between the dependent and independent variables using graphs and tables, and relate these to the equation. For example, in a problem involving motion at constant speed, list and graph ordered pairs of distances and times, and write the equation d = 65t to represent the relationship between distance and time.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Safe-left? Design Recipe (in the student workbook)
- Safe-right? Design Recipe (in the student workbook)
- Onscreen? Design Recipe (in the student workbook)

GETTING STARTED

1) Introduction

Let's get back into that Big Game that we started in stage 7 and continued in stage 12.

When we last worked on the game, our danger and target were moving off the screen in opposite directions. Unfortunately, their update functions move them in one direction forever, so they never come back on screen once they've left! We'd actually like them to have a recurring role in this game, so we'll use some boolean logic to move them back to their starting points once they go off screen.

Once the students correctly implement on-screen? (and its sub-parts safe-left? and safe-right?), the new behavior of target and danger is that once they are off the screen they return to their starting position but with a new y-value. From this new vertical position they will continue to move across the screen. If one (or both) of the characters go off the screen and never reappear, the most likely source of the error is that one of the newly implemented boolean statements is incorrect.

ACTIVITY: THE BIG GAME - BOOLEANS

2) Online Puzzles

Return to your Big Game to use Booleans to keep your player character on screen. Head to CS in Algebra stage 16 in Code Studio to get started programming.

Derived from BOOTSTRAP www.bootstrapworld.org

U

UNPLUGGED

Conditionals and Piecewise Functions

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Currently, even when passing parameters to functions, our outputs follow a very rigid pattern. Now, suppose we want parameters with some values to create outputs using one pattern, but other values to use a different pattern. This is where conditionals are needed. In this stage students will learn how conditional statements can create more flexible programs.

LESSON OBJECTIVES

Students will:

- Understand that piecewise functions evaluate the domain before calculating results.
- Evaluate results of piecewise functions.

ANCHOR STANDARD

Common Core Math Standards

- **F.IF.7.b**: Graph square root, cube root, and piecewise-defined functions, including step functions and absolute value functions.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

Getting Started

1) Vocabulary
2) Conditionals

Activity: Conditionals and Piecewise Functions

3) Conditionals and Piecewise Functions

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Teacher

- Lesson Slide Deck

GETTING STARTED

1) Vocabulary

This lesson has three new and important words:

- **Clause** - a question and its corresponding answer in a conditional expression
- **Conditional** - a code expression made of questions and answers
- **Piecewise Function** - a function which evaluates the domain before choosing how to create the range

2) Conditionals

- We can start this lesson off right away
    - Let the class know that if they can be completely quiet for thirty seconds, you will do something like:
        - Sing an opera song
        - Give five more minutes of recess
        - or Do a handstand
    - Start counting right away.
    - If the students succeed, point out right away that they succeeded, so they *do* get the reward.
    - Otherwise, point out that they were not completely quiet for a full thirty seconds, so they *do not* get the reward.
- Ask the class "What was the *condition* of the reward?"
    - The condition was if you were quiet for 30 seconds
        - If you were, the condition would be true, then you would get the reward.
        - If you weren't, the condition would be false, then the reward would not apply.
    - Can we come up with another conditional?
        - If I say "question," then you raise your hand.
        - If I sneeze, then you say "Gesundheit."
        - What examples can you come up with?

Up to now, all of the functions you've seen have done the same thing to their inputs:

- green-triangle always made green triangles, no matter what the size was.
- safe-left? always compared the input coordinate to 0, no matter what that input was.
- update-danger always added or subtracted the same amount

Conditionals let our programs run differently based on the outcome of a condition. Each clause in a conditional evaluates to a boolean value - if that boolean is TRUE, then we run the associated expression, otherwise we check the next clause. We've actually done this before when we played the boolean game! If the boolean question was true for you, you remained standing, and if it was false you sat down.

Let's look at a conditional piece by piece:

```
(x > 10)  ->  "That's pretty big"
    (x < 10)  ->  "That's pretty small"
    else      ->  "That's exactly ten"
```

If we define x = 11, this conditional will first check if x > 10, which returns TRUE, so we get the String "That's pretty big" - and because we found a true condition we don't need to keep looking.

If we define x = 10, then we first check if x > 10 (FALSE), then we check x < 10 (FALSE), so then we hit the *else* statement, which only returns something if none of the other conditions were true. The *else* statement should be considered the catch-all response - with that in mind, what's wrong with replying "That's exactly ten"? What if x = "yellow"? If you can state a precise question for a clause, write the precise question instead of else. It would have been better to write the two conditions as (x > 10) and (x <= 10). Explicit questions make it easier to read and maintain programs.

Functions that use conditions are called piecewise functions, because each condition defines a separate piece of the function. Why are piecewise functions useful? Think about the player in your game: you'd like the player to move one way if you hit the "up" key, and another way if you hit the "down" key. Moving up and moving down need two different expressions! Without conditionals, you could only write a function that always moves the player up, or always moves it down, but not both.

Now let's play a game.

ACTIVITIES:

3) Conditionals and Piecewise Functions

Living Function Machines - Conditionals:

Explain to the class that they will be playing the role of Function Machines, following a few simple rules: - Whenever your function is called, the only information you are allowed to take in is what's described in your Domain. - Your function must return only what is described in your Range. - You must follow the steps provided in your definition - no magic!

This time, however, everyone will be running the same function. And that function is called 'simon_says' and it has the following Contract: simon_says: String -> Movement
Given a String that describes an action, produce the appropriate movement. If an unknown action is called, lower both hands.

Examples

```
simon_says("left hand up")    = RaiseLeftHand
    simon_says("right hand up")  = RaiseRightHand
    simon_says("left hand down")  = LowerLeftHand
    simon_says("right hand down") = LowerRightHand
```

Definition

```
simon_says(action) = cond {
                "left hand up"     : RaiseLeftHand,
                "right hand up"    : RaiseRightHand,
                "left hand down"   : LowerLeftHand,
                "right hand down"  : LowerRightHand,
                else               : LowerBothHands }
```

Review the contract parts: name, domain, range, parameters (input types), return types (output values)

Say to the class: “Here is what the initial code looks like. We will add several clauses but the clauses that are there will always be there and the final else action (often called the default result) will always be LowerBothHands

- simon_says("right hand up")
- simon_says("left hand up") - both hands should be up
- simon_says("right hand up") - both hands should still be up
- simon_says("left hand down") - left should be down, right should be up
- simon_says("right hand up") - left should be down, right should be up
- simon_says("hokey pokey") - both hands should be down
- simon_says("left hand up") - left hand should be up
- simon_says("right up") - trick, there are no matches so the else statement is called

If anyone makes a mistake, they must "reboot" by sitting down and waiting for the next round to start.

Say to the class: “Now we're going to rewrite our function a little bit - instead of taking a String as its Domain, simon_says will take a Number. Here's what our new function looks like:

```
simon_says(action) = cond {
```

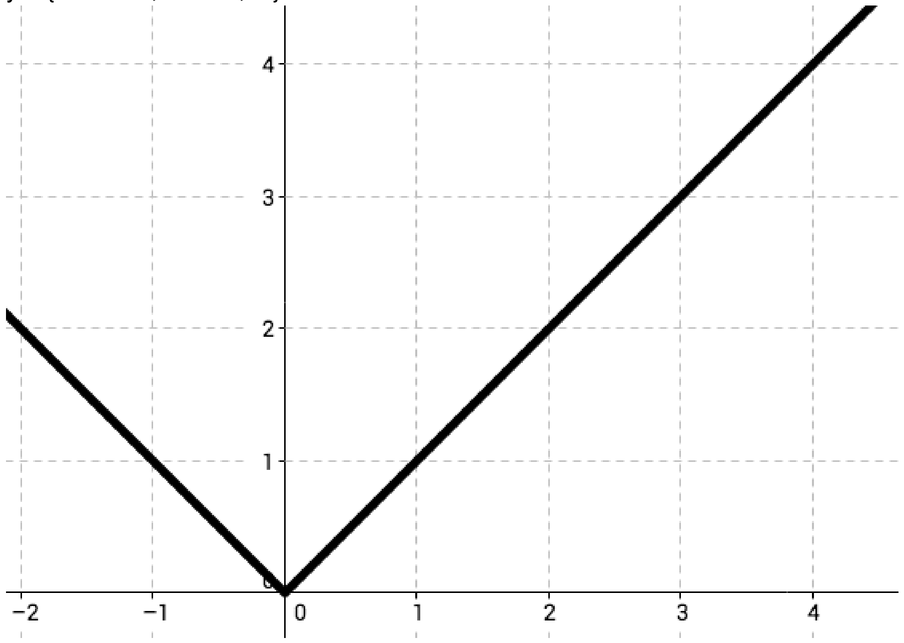
```
(action < 10)                  : RaiseLeftHand,
(action < 20)                  : RaiseRightHand,
(action > 20) and (action < 50)  : LowerLeftHand,
(action > 50) and (action < 100) : LowerRightHand,
else                        : LowerBothHands }
```

Continue playing using numbers in the simon_says function, such as simon_says(15), which should result in RaiseRightHand. As students get comfortable with the new rules, you can throw in some trick questions, such as simon_says(20) or simon_says(50), both of which should call the else statement. You can extend this activity in many ways, for example:

- Call the function with a simple expression, such as simon_says(30 / 2)
- Add more conditions of your own
- Create multiple functions and divide the class into groups
- Allow students to take over as the 'programmer'

Connection to Mathematics and Life

There are piecewise functions in mathematics as well. The absolute value function y = |x| can be re-written as y = { -x : x<0 , x : x>0, 0 }

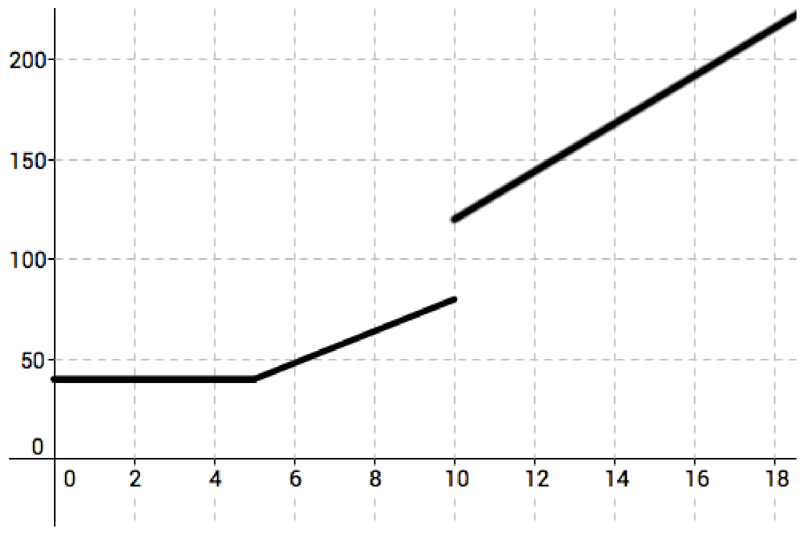


Note that in mathematical terms, the clause for the domain is usually listed second instead of first.

A data plan on a phone bill might be structured as:

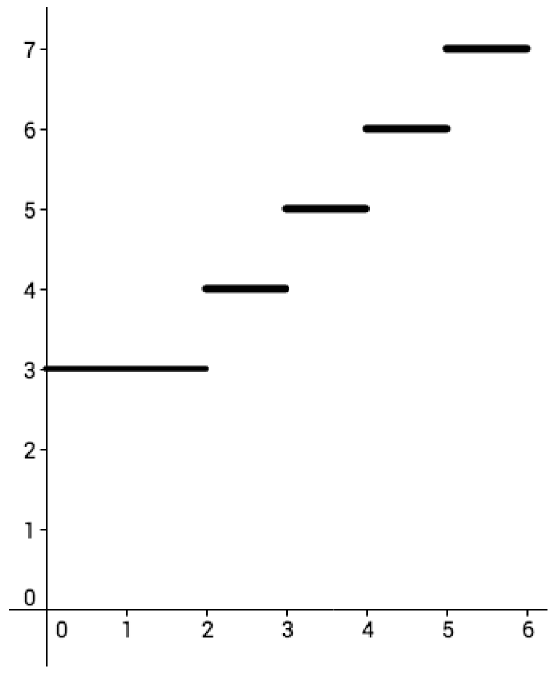
- $40 for less than 5 GB
- $ 8 per GB for 5-10 GB
- $12 per GB for using more than 10GB

This could be graphed with the following piecewise function y = { 40: x<5, 8x: 5 =< x =< 10, 12x: x>10 }



Another very common piecewise functions is for taxi cabs.

- $3 for 0 to 2 miles
- $1 for each partial mile after that



This could be graphed with the following piecewise function y = { 3: x<2, [[x]]+2: x>=2 } where [[x]] is the greatest integer function or what is often called a floor function in computer languages. The greatest integer function returns the greatest INTEGER less that the current value. For instance [[2.9]] is 2 and [[3.1]] is 3.

Derived from
BOOTSTRAP
www.bootstrapworld.org

18

CS in Algebra | Lesson 18

Conditionals and Update Player

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using conditionals, students will write functions and programs that change their behavior based on logical evaluation of input values.

LESSON OBJECTIVES

Students will:

- Use Boolean operators to compare values.
- Apply Boolean logic, such as AND, OR, and NOT, to compose complex Boolean comparisons.
- Write conditional statements that evaluate differently based on their input values.

ANCHOR STANDARD

Common Core Math Standards

- **6.NS.8**: Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

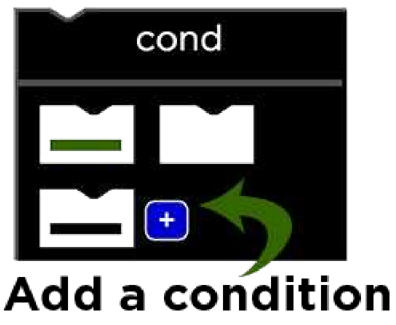
- Cost Design Recipe in the student wor boo)
- Update-player Design Recipe (in the student workbook)
- Key Code Reference (in the student workbook)

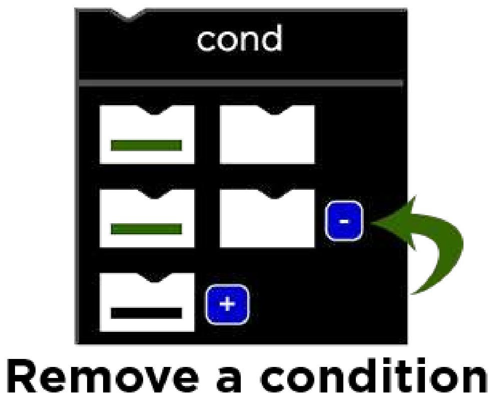
GETTING STARTED

1) Introduction

Remind students of the game they played in the last stage. What were some of the tricky elements of constructing a good conditional statement?

- Order matters (the first condition in the list to return true wins).
- Write clear and explicit conditions.
- Use the else clause as a catch-all for conditions that you don't expect or can't write explicit conditions for.
- All conditionals must have at least one condition and an else statement, you can add or remove further conditions using the blue buttons.





At the end of this stage, students will return to their Big Game to complete the update-player function. This function contains a conditional that will check which key was pressed (using key codes), and move the player up or down accordingly. We've provided a key code reference for students in case they wish to use keys other than the default up (38) and down (40) arrows.

LESSON TIP

Be sure to check students' Contracts and Examples during this exercise, especially when it's time for them to circle and label what changes between examples. This is the crucial step in the Design Recipe where they should discover the need for cond.

ACTIVITY: CONDITIONALS

2) Online Puzzles

Head to CS in Algebra stage 18 in Code Studio to get started programming.

EXTENSION ACTIVITIES

3) Improving Luigi's Pizza

The final puzzle in the Luigi's Pizza sequence is a Free Play puzzle that allows for students to extend the program in a number of different ways. While some of the potential extensions seem simple, they can be deceptively challenging to get working. Allow students to explore extensions individually, or choose one to work through as a whole class.

- **Coupon Code**: Write a function coupon that takes in a topping and a coupon code and returns the price of a pizza with that topping, with %40 off is the code is correct.

- **Multiple Toppings**: Write a function two-toppings that takes in two toppings and returns the price of a pizza with those toppings.
- **Picture Menu**: Write a function pizza-pic that takes in a topping and returns a simple image representing a pizza with that topping.

3) Update Player

The update-player function is one of the most extensible in the Big Game. Here's a brief list of potential challenge extensions to give students:

- **Warping**: instead of having the player's y-coordinate change by adding or subtracting, replace it with a Number to have the player suddenly appear at that location. (For example, hitting the "c" key causes the player to warp back to the center of the screen, at y=200.)
- **Boundary-detection**: Keep the player on screen by changing the condition for moving up so that the player only moves up if the up ke was pressed AND player-y is below the top border. Likewise, change the condition for down to also check that player-y is above the bottom.
- **Wrapping**: Add a condition (before any of the keys) that checks to see if the player's y-coordinate is above the screen. If it is, have the player warp to the bottom. Add another condition so that the player warps back up to the top of the screen if it moves below the bottom.
- **Dissapear/Reappear**: Have the player hide when the "h" key is pressed, only to re-appear when it is pressed again!

Derived from BOOTSTRAP www.bootstrapworld.org

UNPLUGGED

Collision Detection and the Pythagorean Theorem

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Determining when objects on the screen touch is an important aspect of most games. In this lesson we'll look at how the Pythagorean Theorem and the Distance Formula can be used to measure the distance between two points on the plane, and then decide whether those two points (or game characters) are touching.

LESSON OBJECTIVES

Students will:

- Demonstrate that circles will overlap if the distance between their centers is less than the sum of their radii.
- Show that the distance of two points graphed in 2 dimensions can be represented as the hypotenuse of a right triangle.
- Understand that the Pythagorean Theorem allows you to calculate the hypotenuse of a right triangle using the length of the two legs.
- Apply the Pythagorean Theorem to calculate the distance between the centers of two objects.

ANCHOR STANDARD

Common Core Math Standards

- **8.G.7**: Apply the Pythagorean Theorem to determine unknown side lengths in right triangles in real-world and mathematical problems in two and three dimensions.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- ollision Worksheets
- Safe-right? Design Recipe (in the student workbook)
- Onscreen? Design Recipe (in the student workbook)

For the Teacher

- Language Table (see below)
- Cutouts of Pythagorean Theorem packets (1, 2) - 1 per group of students working together

GETTING STARTED

1) Vocabulary

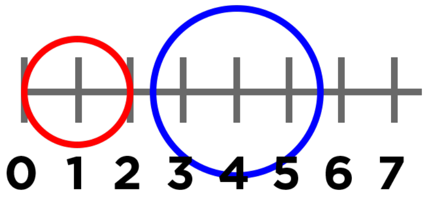
This lesson has one new and important word:

- **Hypotenuse** - the side opposite the 90-degree angle in a right triangle

2) Are they Touching?

Suppose two objects are moving through space, each one having its own (x,y) coordinates. When do their edges start to overlap? They certainly overlap if their coordinates are identical ($x_1 = x_2$, $y_1 = y_2$), but what if their coordinates are separated by a small distance? Just how small does that distance need to be before their edges touch?

Visual aids are key here: be sure to diagram this on the board! In one dimension, it's easy to calculate when two objects overlap. In this example, the red circle has a radius of 1, and the blue circle has a radius of 1.5 The circles will overlap if the distance *between their centers* is *less than the sum of their radii* (1 + 1.5 = 2.5). How is the distance between their centers calculated? In this example, their centers are 3 units apart, because 4 – 1 = 3.



LESSON TIP

Would the distance between them change if the circles swapped places? Why or why not?

Work through a number of examples, using a number line on the board and asking students how they calculate the distance between the points. Having students act this out can also work well: draw a number line, have two students stand at different points on the line, using their arms or cutouts to give objects of different sizes. Move students along the number line until they touch, then compute the distance on the number line.

Your game file provides a function called line-length that computes the difference between two points on a number line. Specifically, line-length takes two numbers as input and determines the distance between them

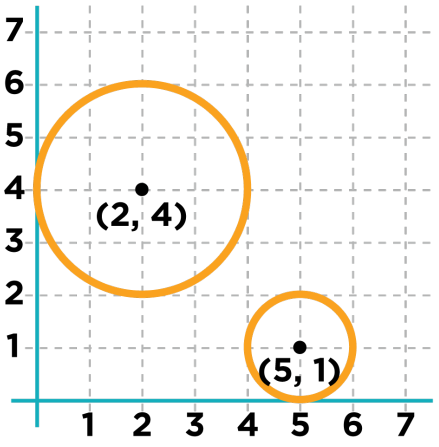
LESSON TIP

What answers would you expect from each of the following two uses of line-length:

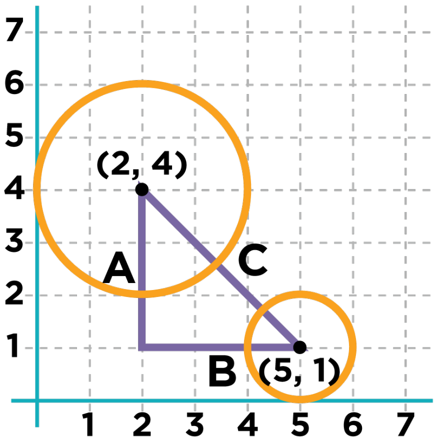
- line-length(2, 5)
- line-length(5, 2)

Do you expect the same answer regardless of whether the larger or smaller input goes first?

Unfortunately, line-length can only calculate the distance between points in a single dimension (x or y). How would the distance be calculated between objects moving in 2-dimensions (like your game elements)? **line-length** can calculate the vertical and horizontal lines in the graphic shown here, using the distance between the x-coordinates and the distance between the y-coordinates. Unfortunately, it doesn't tell us how far apart the two centers are.



Drawing a line from the center of one object to the other creates a right-triangle, with sides A, B and C. A and B are the vertical and horizontal distances, with C being the distance between the two coordinates. **line-length** can be used to calculate A and B, but how can we calculate C?



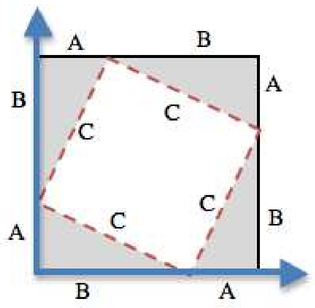
In a right triangle, the side opposite the 90-degree angle is called the hypotenuse. Thinking back to our collision detection, we know that the objects will collide if the hypotenuse is less than the sum of their radii. Knowing the length of the hypotenuse will be essential to determine when a collision occurs.

ACTIVITIES:

3) Proving Pythagoras

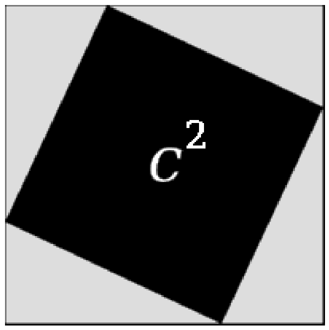
If your students are new to the Pythagorean Theorem, or are in need of a refresher, this activity is an opportunity to strengthen their understanding in a hands-on fashion.

Organize students into small groups of 2 or 3.



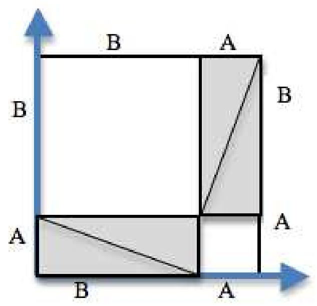
- Pass out Pythagorean Proof materials (1, 2) to each group.
- Have students cut out the four triangles and one square on first sheet.
- Explain that, for any right triangle, it is possible to draw a picture where the hypotenuse is used for all four sides of a square.
- Have students lay out their gray triangles onto the white square, as show in this diagram.
- Point out that the square itself has four identical sides of length C, which are the hypotenuses for the triangles. If the area of a square is expressed by *side ∗ side*, then the area of the white space is C^2.
- Have students measure the inner square formed by the four hypotenuses (C)

By moving the gray triangles, it is possible to create two rectangles that fit inside the original square. While the space taken up by the triangles has shifted, it hasn't gotten any bigger or smaller. Likewise, the white space has been broken into two smaller squares, but in total it remains the same size. By using the side-lengths A and B, one can calculate the area of the two squares.



- You may need to explicitly point out that the side-lengths of the triangles can be used as the side-lengths of the squares.
- Have students measure the area of the smaller square (A)
- Have students measure the area of the larger square (B)
- Ask students to compare the are of square A + square B to the area of square C

The smaller square has an area of A^2, and the larger square has an area of B^2. Since these squares are just the original square broken up into two pieces, we know that the sum of these areas must be equal to the area of the original square:



$$A^2 + B^2 = C^2$$

4) Collision Detection

In this activity students will:

- Create right triangles on a graph.
- Calculate the hypotenuse by direct measurement and by the Pythagorean Theorem.
- Determine if circles have collided by examining visually.
- Determine if circles have collided by comparing distance and radii.

Detailed instructions are provided on the Collision Worksheet.

Derived from BOOTSTRAP www.bootstrapworld.org

20

CS in Algebra | Lesson 20

The Big Game - Collision Detection

Lesson time: 30-60 Minutes

LESSON OVERVIEW

To finish up their video games, students will apply what they have learned in the last few stages to write the final missing functions. We'll start by using booleans to check whether keys were pressed in order to move the player sprite, then move on to applying the Pythagorean Theorem to determine when sprites are touching.

LESSON OBJECTIVES

Students will:

- Apply the Distance Formula to detect when two points on a coordinate plane are near each other.

ANCHOR STANDARD

Common Core Math Standards

- **8.G.7**: Apply the Pythagorean Theorem to determine unknown side lengths in right triangles in real-world and mathematical problems in two and three dimensions.

Additional standards alignment can be found at the end of this lesson

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Line-length Design Recipe (in the student workbook)
- Distance Design Recipe (in the student workbook)
- Collide? Design Recipe (in the student workbook)

GETTING STARTED

1) Introduction

Let's get back into that Big Game from stages 7, 12, and 16.

Previous work with the game has created movement for the danger and target characters, using Booleans to check if they have left the screen. The last time students worked on their game they used a conditional to check which key was pressed and make the player move accordingly. At this point the only thing left to do is to decide when the player is touching either the target or danger. Once students have successfully completed the **distance** and **collide?** functions, their score will increase when the player touches the target, and decrease when it touches the danger.

The Pythagorean Theorem studied in the last lesson will be used to determine when the characters have made contact. Students are not required to write their own line-length function, but you may ask them to complete the Design Recipe for it anyway.

Students will first complete the distance function so that it measures the distance between two points, (px, py) and (cx, cy). After the students implement the distance formula, they will need to implement the tests in the collide? function.

Once these last functions are put into place, scoring will automatically update based on collisions between target and danger.

ACTIVITY: THE BIG GAME - COLLISION DETECTION

2) Online Puzzles

Return to your Big Game to use collision detection logic so that you know when your player is touching the target or the danger. Head to CS in Algebra stage 20 in Code Studio to get started programming.

Derived from BOOTSTRAP www.bootstrapworld.org