Lessons Used in This Pathway

The Numbers Inside Video Games

The Numbers Inside Video Games

(Also available for Pyret)

Students reverse engineer a video game and research what it takes to create a video game.

Lesson Goals	Students will be able to:

	• Identify the objects in a video game that are changing. • Use math language to describe what is changing about each object. • Understand the time, money, and resources it takes to create a popular video game.
Student-Facing Lesson Goals	• I can identify the objects in a video game. • I can use math vocabulary to describe what is changing about each object. • I understand the time, money, and resources it takes to create a popular video game.
Materials	• Lesson Slides • NinjaCat demo game • Notice and Wonder (Page 2) • Reverse Engineer a Video Game (Page 3)
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Key Points for the Facilitator	• Students will need their own Google accounts. • Take care to manage student expectations about what their game will be like. Modern games are very complex!
Supplemental Resources	• What's going on in this Graph?

Click here to see the prior unit-based version.

Reverse Engineering a Video Game

25 minutes

Overview

Students play a simple video game, and gradually break it down into parts. Doing so reveals how coordinates play a crucial role in video games, and how animation is created via equations that govern the changing values of those coordinates.

Launch

Play the NinjaCat demo game onscreen while students watch. Purposely make mistakes while playing the game, which should elicit responses/direction from students.

> Take turns playing the game in pairs. After you’ve both had a chance to play, write down what you *notice* about the game on Notice and Wonder (Page 2). "Notice"s should be statements, not questions - What stood out to you? What do you remember?

Crowdsource students' Notices.

> What do you *wonder* about the game? What questions do you have about how it works? Write these down on Notice and Wonder (Page 2).

Crowdsource students' Wonders.

Pedagogy Note!

This pedagogy has a rich grounding in literature, and is used throughout this course. In the "Notice" phase, students are asked to crowd-source their observations. No observation is too small or too silly! By listening to other students' observations, students may find themselves taking a closer look at the game. The "Wonder" phase involves students raising questions, but they must also explain the context for those questions. Sharon Hessney (moderator for the NYTimes excellent What's going on in this Graph? activity) sometimes calls this "what do you wonder…and **why**?". Both of these phases should be done in groups or as a whole class, with adequate time given to each.

Investigate

Students complete the Reverse Engineer a Video Game (Page 3) worksheet in pairs.

- 1st Column: What are all the various *things* in this game? *(A dog, Clouds, etc.)*
- 2nd Column: For each of those "things", what is changing about them? *(Location, Position, etc.)*
- 3rd Column: For each change, how is it modeled mathematically? *(x-coordinate, y-coordinate, amount, etc.)*

Common Misconceptions

- Students are likely to describe what the character is *doing*, as opposed to *what changes*. For example: "The dog is moving to the left" is not actualy describing the property being changed (position, place, location, etc).
- Students may write down what they *hope* is changeable, as opposed to what actually changes. It's common for students to say they cat's costume is changing, because they assume the cat will somehow "level up" if they get enough points.

Synthesize

The main idea here is to understand that while we see images on a screen, the computer only sees a small set of numbers, which uniquely model the state of the game. The way those numbers change determines how the game behaves, and we can add features to the game if we’re willing to keep track of more numbers.

- If the x- and y-coordinates are each numbers, how many numbers does it take to represent a single frame of the video game?
- How are those numbers changing - or *varying* - as the game plays? When do they increase? Decrease?
- How many numbers would we need if the dog could also move up and down?
- How many numbers would we need to have a two-player game?
- How many numbers would we need if the entire game was in 3d?
- How many numbers would we need to make a modern game?

Connecting to Real Games 25 minutes

Overview

Students apply this way of thinking to more complex, real-world games. They also get a sense for how much work is involved in creating games like that.

Launch

Ask students to share out their favorite current video game. Write the names of the games on the board.

Investigate

Let students choose a current, popular game to discuss.

Collect students estimates for each of the questions below.

- How long do you think it took to create that game?
- How *many people* do you think it takes to create a game like that?
- How *much money* does it take to create a game like that?

Optional: Once students have made their estimates, have students use the Internet to research these questions and compare the actual numbers to their estimates.
The goal here is not to discourage students from the possibility of eventually creating a game like their favorite game, but to manage expectations given our limited resources (time, money, and people). By starting with this game project, students are learning transferable skills that can help them later on in learning new programming languages and creating bigger projects.

Synthesize

- Share-back: have students share their estimates with the class. Was anything drastically higher or lower than they expected?
- What does this tell us about making modern games?
- Are we likely to create games like the ones you researched?

The 3d, two-player version of NinjaCat needed a lot more numbers than the simple one you saw here, *but the actual concepts at work are the same* . Even if we don’t have time to make games like the ones we chose here, you’ll learn the same concepts just by making a simpler one.

Coordinates and Game Design

Coordinates and Game Design

[(Also available for Pyret)](#)

Students review the importance and need for coordinates in the context of a video game and brainstorm a game of their own.

Lesson Goals	Students will be able to: • Explain the need for ***coordinates*** in a given situation. • Estimate coordinates in a bounded area.
Student-Facing Lesson Goals	• I can estimate the positions of objects using coordinates. • I can collaborate with a partner to brainstorm a video game. • I can create a sample mock-up (proof of concept) of my video game.
Materials	• [Lesson Slides](#) • [Estimating Coordinates (Page 4)](#) • [Brainstorm Your Own Game (Page 5)](#)

	• Ninja Cat Desmos graph (Desmos) • Google Draw template • Optional: cutouts of the Cat, Dog, and Ruby from the NinjaCat game.
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Key Points for the Facilitator	• The launch activity should create and reinforce the need for coordinates and to attend to precision. • Continue to use the same "Estimating Coordinates" page so students can track their pattern of estimation over time.

Click here to see the prior unit-based version.

Glossary

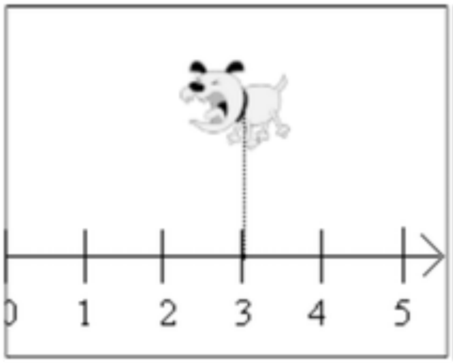
coordinate :: a number or set of numbers describing an object’s location
horizontal axis :: axis on a coordinate plane that runs from left to right
vertical axis :: number line on a coordinate plane that runs from bottom to top, indicating values in that direction

Navigating a Grid 20 minutes

Overview

Students are asked to come up with a way of identifying location on a grid, which provides the justification for coordinates.

Launch



Computers use numbers to represent a character's position onscreen, using number lines as rulers to measure the distance from the bottom-left corner of the screen. For our videogame, we will draw the number line so that the screen runs from 0 (on the left) to 1000 (on the right).

We can take the image of the Dog, 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 Dog, as long as they know the number.

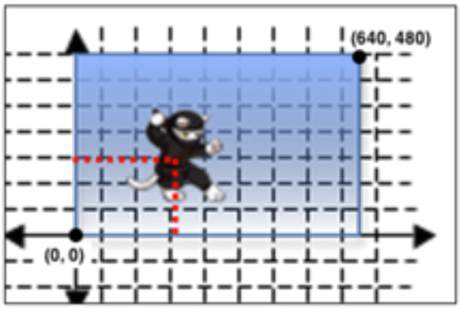
- What is the coordinate of the Dog, if it's on the left-hand edge of the screen?
- What is the coordinate of the Dog, if it's on the right-hand edge of the screen?
- What is the coordinate of the Dog, if it's in the center of the screen?
- What coordinate would place the Dog beyond the left-hand edge of the screen?
- What coordinate would place the Dog beyond the right-hand edge of the screen?

OPTIONAL: Draw a number line on the board, and select a volunteer to leave the room for a moment. Place the printed Dog image somewhere on that line, and have the class quietly choose the number that represents the Dog's location. Remove the Dog and invite the student back into the room. Can they position the Dog at the right place, based on the number chosen by the class?

This number line lets us communicate the position of the Dog using a single number! Unfortunately, it only represents the distance from the left-hand edge of the screen. That means the dog could be at any *height* in the center of the screen, and it would always have the same number to represent its position.

Investigate

By adding a second number line, we can locate a character *anywhere* on the screen in either direction. The first line we drews 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 NinjaCat's position on the screen. We can find the x-coordinate by dropping a line down from NinjaCat and read the position on the number line. The y-coordinate is found by running a line to the y-axis.

A coordinate pair is always written in the form of (x, y). 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.

To develop an intuition for coordinates, have students complete Estimating Coordinates (Page 4).

Common Misconceptions

Math-phobic students often fail to realize that *common sense* and *intuition* can be helpful in exercises where the answer is a number! The first two prompts in the "Synthesize" section directly get at this misconception, but you may want to pay special attention to those students while they are working on this workbook page.

Synthesize

- Should any of the characters have x-coordinates that are very similar? How come?
- Should any of the characters have y-coordinates that are very similar? How come?
- How do you think this concept relates to a video game? *Answers vary: we need to know where characters are on the screen, we need a way for players to interact with certain parts of the screen, etc*

Bridging to video games 30 minutes

Overview

Students explore a coordinate activity in which a cartesian point is used to compute the position of a character in a game. From there, they brainstorm a game of their own.

Launch

In pairs, have students explore the Ninja Cat Desmos graph (Desmos).

Investigate

- Students complete the Brainstorm Your Own Game (Page 5) worksheet and decide on a Player, Target, Danger, and Background for their game.
- Students will use a Google Draw template (click "Make a copy" when prompted) to create a sample "screenshot" of their game by inserting images via Google Search.

Screenshot should include:

- Labeled estimates of coordinates for each character.
- 2 characters that have the same x-coordinate.
- 2 different characters that have the same y-coordinate.

Synthesize

- When the "Game Over" screen is supposed to be off screen, what coordinates might hide it?
- What would be the coordinate of the dog *before it gets onscreen?*
- Why do we estimate? *Practice number sense, get better at working with numbers*
- What constitutes a good estimate?
- How can we improve our estimation skills? *Practice, get more comfortable with numbers and more comfortable with making guesses*

Order of Operations

Order of Operations

(Also available for Pyret)

Students learn to model arithmetic expressions with a visual tool for Order of Operations, known as "Circles of Evaluation".

Lesson Goals	Students will be able to: • Model an arithmetic expression using ***Circles of Evaluation***. • Translate Circles of Evaluation into code.
Student-facing Goals	• I can write Circles of Evaluation for a given arithmetic ***expression***. • I can translate a Circle of Evaluation model into code. • I can use numbers and operations in a programming environment.
Materials	• Lesson Slides • Completing Circles of Evaluation from Arithmetic Expressions (Page 7) • Matching Circles of Evaluation and Arithmetic Expressions (Page 8) • Translate Arithmetic to Circles of Evaluation & Code (Intro) (Page 9) • Completing Partial Code from Circles of Evaluation (Page 10) • Matching Circles of Evaluation & Code (Page 11)

	• Translate Arithmetic to Circles of Evaluation & Code 2 (Page 12) • Arithmetic Expressions to Circles of Evaluation & Code - Challenge (Page
Preparation	• Make sure all materials have been gathered
Key Points For The Facilitator	• Error messages are the computer trying to give us a clue that something is wrong. Model reacting to ***error messages*** with interest to demonstrate to students that the messages are a helpful tool. • After the first few exercises in creating Circles of Evaluation, ask students whether they create them from the 'inside-out' (drawing the inner circles first) or from the 'outside-in.' After they've given their responses, have them try using the OTHER way! • Up until now, we didn't have a visual spatial model for *reading* arithmetic expressions. Ask students to compare Circles of Evaluation to previous methods they've learned for *computing* these expressions (PEMDAS, GEMDAS, etc) • For a memory hook, model the "bug that crawls through the circle" explanation. • Students may benefit from using multiple colors to distinguish between the different smaller expressions and parentheses.

Click here to see the prior unit-based version.

Glossary

circle of evaluation :: a 'sentence diagram' of the structure of a mathematical expression

error message :: information from the computer about errors in code

expression :: a computation written in the rules of some language (such as arithmetic, code, or a Circle of Evaluation)

function :: a mathematical object that consumes inputs and produces an output

Order of Operations 30 minutes

Overview

Students are given a challenging expression that exposes common misconceptions about order of operations. The goal is to demonstrate that a brittle, fixed notion of order of operations is insufficient, and lead students to a deeper understanding of order of operations as a grammatical device. The Circles of Evaluation are introduced as "sentence diagramming for arithmetic".

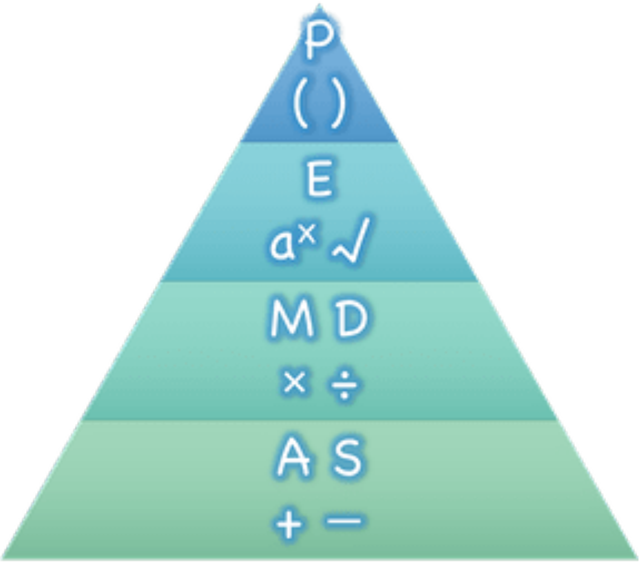
Launch

Humans use verbs like "throw", "run", "build" and "jump" to describe operations on nouns like "ball", "puppy", and "blocks". Mathematics has "operations" and ***functions***, like addition and subtraction. Just as you can " **throw** a *ball* ", a person can also " **add** *four* and *five* ".

A mathematical expression is an instruction for doing something, which specifies the verbs and nouns involved. The expression $4 + 5$ tells us to add 4 and 5. To evaluate an expression, we follow the instructions. The expression $4 + 5$ evaluates to 9.

If you were to write instructions for getting ready for school, it would matter very much which instruction came first: putting on your socks, putting on your shoes, etc. Sometimes we need multiple expressions in mathematics, and the order matters there, too!

Mathematicians didn't always agree on the order of operations, but at some point it became important to develop rules to help them work together.



The pyramid on the right is a model for summarizing the order of operations. When evaluating an expression, we begin by applying the operations written at the top of the pyramid (operations in parentheses and other grouping symbols). Only after we have completed all of those operations can we move down to the lower level. If both operations from the same level 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. This set of rules is brittle, and doesn't always make it clear what we need to do. Mnemonic devices for the order of operations like PEMDAS, GEMDAS, etc focus on how to get the answer. What we need is a *better way to read math* .

Check out the expression below. What do you think the answer is? This math problem went viral on social media recently, with math teachers arguing about what the answer was! Why might they disagree on the solution?

$$6 \div 2(1 + 2)$$

Instead of using a rule for computing answers, let's start by diagramming the math itself!

Circles of Evaluation

The Circles of Evaluation are a critical pedagogical tool in this course. They place the focus on the *structure* of mathematical expressions, as a means of combating the harmful student belief that the only thing that matters is the *answer*. They can be used to diagram arithmetic sentences to expose common misconceptions about Order of Operations, and make an excellent scaffold for tracing mistakes when a student applies the Order of Operations incorrectly. They are also a bridge representation, which naturally connects to function composition and converting arithmetic into code.

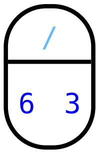
We can *draw the structure* of this grammar in mathematics using something called the **Circles of Evaluation**. The rules are simple:

1) Every Circle must have one - and only one! - function, written at the top.

That means that Numbers (e.g. - $3, -29, 77.01$...) are still written by themselves. It's only when we want to *do something* like add, subtract, etc. that we need to draw a Circle.

2) The inputs to the function are written left-to-right, in the middle of the Circle.

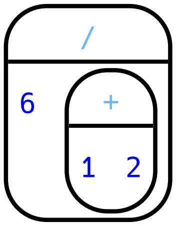
If we want to draw the Circle of Evaluation for $6 \div 3$, the division function (`/`) is written at the top, with the `6` on the left and the `3` on the right.



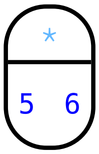
What if we want to use multiple functions? How would we draw the Circle of Evaluation for $6 \div 1 + 2$? Drawing the Circle of Evaluation for the $1 + 2$ is easy. But how do we take the result of that circle, and divide 6 by it?

Circles can contain other Circles

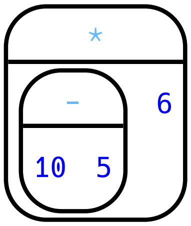
We basically replace the `3` from our earlier Circle of Evaluation with *another* Circle, which adds 1 and 2!



What would the Circle of Evaluation for 5×6 look like?



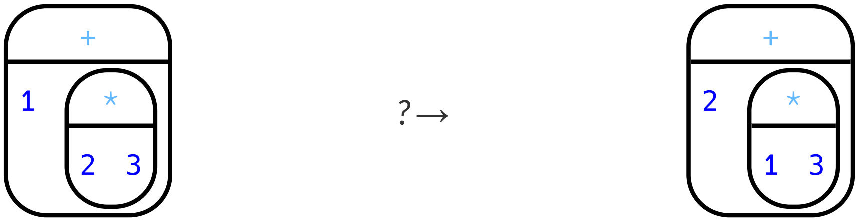
How about the Circle of Evaluation for $(10 - 5) \times 6$?



Aside from helping us catch mistakes before they happen, Circles of Evaluation are also a useful way to think about *transformation* in mathematics. For example, you may have heard that "addition is commutative, so $a + b$ can always be written as $b + a$." For example, $1 + 2$ can be transformed to $2 + 1$.

Suppose another student tells you that $1 + 2 \times 3$ can be rewritten as $2 + 1 \times 3$. This is obviously wrong, but *why*?

Take a moment to think: what's the problem? We can use the Circles of Evaluation to figure it out! The first Circle is just the original expression. The second expression represents what the (incorrect) commutativity transformation gives us:



In this case, the student *failed to see the structure*, viewing the term to the right of the $+$ sign as 2 instead of 2×3. The Circles of Evaluation help us see the structure of the expression, rather than forcing us to construct it and keep it in our heads.

Investigate

Have students turn to Translate Arithmetic to Circles of Evaluation & Code (Intro) (Page 9) in the student workbook and draw Circles of Evaluation for each of the expressions. (Ignore the code column for now! We will come back to it later.)

You may also want to have students complete Completing Circles of Evaluation from Arithmetic Expressions (Page 7), Matching Circles of Evaluation and Arithmetic Expressions (Page 8) and/or Matching Circles of Evaluation to Expressions (Desmos).

Pedagogy Note

Circles of Evaluation are a great way to get older students to reengage with (and finally understand) the order of operations while their focus and motivation are on learning to code. Because we recognize this work to be so foundational, and know that some teachers choose to spend a whole week on it, we have developed lots of additional materials to help scaffold and stretch. You will find some additional pages in the workbook and over 20 more linked in the Additional Exercises section at the the end of this lesson.

Synthesize

- Did some students prefer working outside-in to inside-out? Why?
- Did some students find that different strategies worked better for different *kinds* of problems? Why or why not?
- Is there more than one way to draw the Circle for $1 + 2$? If so, is one way more "correct" than the other?

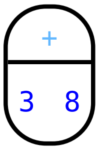
From Circles of Evaluation to Code

Overview

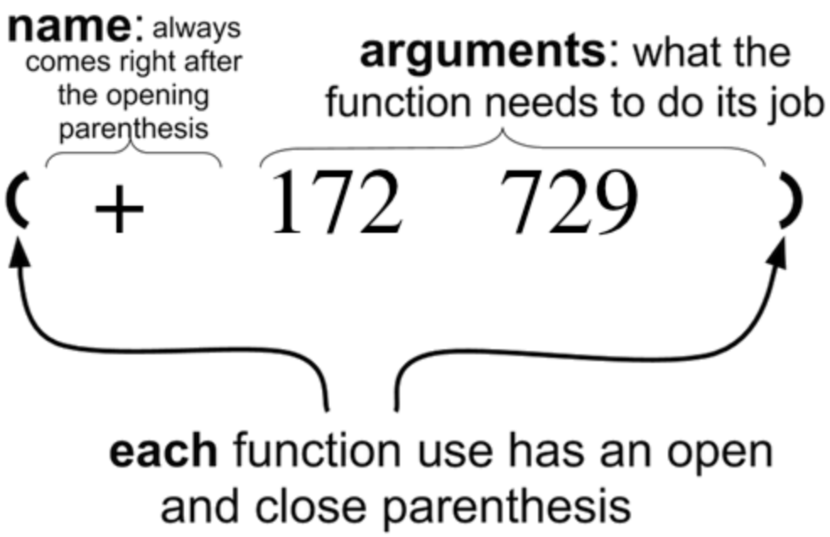
Students learn how to use the Circles of Evaluation to translate arithmetic expressions into code.

Launch

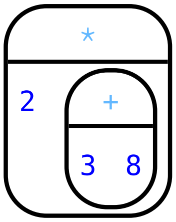
When converting a Circle of Evaluation to code, it’s useful to imagine a spider crawling through the circle from the left and exiting on the right. The first thing the spider does is cross over a curved line (an open parenthesis!), then visit the operation - also called the *function* - at the top. After that, she crawls from left to right, visiting each of the inputs to the function. Finally, she has to leave the circle by crossing another curved line (a close parenthesis).

Expression	→	$3 + 8$
Circle of Evaluation	→	
Code	→	`(+ 3 8)`

All of the expressions that follow the function name are called arguments to the function. The following diagram summarizes the shape of an expression that uses a function.

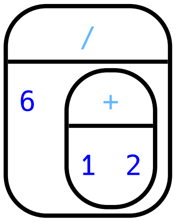

name: always comes right after the opening parenthesis
arguments: what the function needs to do its job
(+ 172 729)
each function use has an open and close parenthesis

Arithmetic expressions involving more than one operation, will end up with more than one circle, and more than one pair of parentheses.

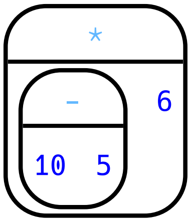
Expression	→	$2 \times (3 + 8)$
Circle of Evaluation	→	
Code	→	`(* 2 (+ 3 8))`

- Why are there two closing parentheses in a row, at the end of the code?
- If an expression has three sets of parentheses, how many Circles of Evaluation do you expect to need?

What would the code look like for these circles?



```
(/ 6 (+ 1 2))
```



```
(* (- 10 5) 6)
```

Investigate

If you have time, start with the two pages in the student workbook that scaffold translating circles to code: Completing Partial Code from Circles of Evaluation (Page 10) and Matching Circles of Evaluation & Code (Page 11).

Now that we know how to translate Circles of Evaluation into Code, turn back to Translate Arithmetic to Circles of Evaluation & Code (Intro) (Page 9).

> **Before you have students complete the code for this page, make sure they have drawn their circles correctly!** You may want to have them compare their circles with a partner and another pair of partners or you may want to post an answer key.

Once you confirm that your code is correct, continue on to Translate Arithmetic to Circles of Evaluation & Code 2 (Page 12)

> (The previous workbook page scaffolded students' in translating expression to code with extra parentheses. Those scaffolds drop away on this page.)

If time allows, take turns entering the code into the editor with your partner.

We have included one page of more complex problems in the student workbook so that you're ready to challenge students who fly Arithmetic Expressions to Circles of Evaluation & Code - Challenge (Page 13).

Note: If you want to practice making Circles of Evaluation with exponents and square roots, we use `sqrt` as the name of the square root function, and `sqr` as the function that squares its input.

Strategies For English Language Learners

MLR 7 - Compare and Connect: Gather students' graphic organizers to highlight and analyze a few of them as a class, asking students to compare and connect different representations.

Synthesize

Have students share back what they learned from the Circles of Evaluation.

Testing out your Code optional

Overview

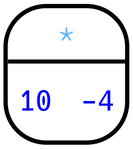
Circles of Evaluation are a powerful tool that can be used without ever getting students on computers. If you have time to introduce students to the wescheme editor, typing their code into the interactions area gives students a chance to get feedback on their use of parentheses as well as the satisfaction of seeing their code successfully evaluate the expressions they've generated.

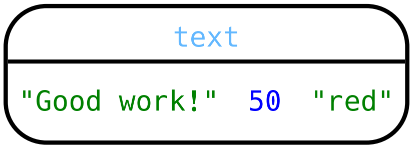
Launch

- Open WeScheme and click run.
- For now, we are only going to be working in the interactions area on the right hand side of your screen.
- Type `(+ (* 8 2) (/ 6 3))` into the interactions area.
- Notice how the editor highlights pairs of parentheses to help you confirm that you have closed each pair.
- Hit Enter (or Return) to evaluate this expression. What happens? *If you typed the code correctly you'll get 18. If you make a mistake with your typing, the computer should help you identify your mistake so that you can correct it and try it again!*
- Take a few minutes to go back and test each line of code you wrote on the pages you've completed by typing them into the Interactions Area. Use the error messages to help you identify any missing characters and edit your code to get it working.

Investigate

Here are two Circles of Evaluation.



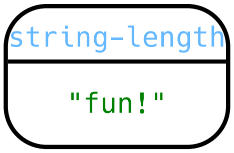


One of them is familiar, but the other is very different from what you've been working with. What's different about the Circle on the right?

Possible responses:

  - *We've never seen the function* `text` *before*
  - *We've never seen words used in a Circle of Evaluation before*
  - *We've never seen a function take in three inputs*
  - *We've never seen a function take in a mix of Numbers and words*
- Can you figure out the Name for the function in the second Circle? This is a chance to look for and make use of structure in deciphering a novel expression! *We know the name of the function is* `text`*, because that's what is at the top of the circle.*
- What do you think this expression will evaluate to?
- Convert this Circle to code and try it out!
- What does the `50` mean to the computer? Try replacing it with different values, and see what you get.
- What does the `"blue"` mean to the computer? Try replacing it with different values, and see what you get.

Here is another circle to explore.



- What do you think this expression will evaluate to?
- Convert this Circle to code and try it out!

Synthesize

Now that we understand the structure of Circles of Evaluation, we can use them to write code for any function!

Additional Exercises

If you are digging into Order or Operations and are looking for more practice with Circles of Evaluation before introducing code, we have lots of options!

- A printable set of cards for physically matching expressions with Circles of Evaluation
- Creating Circles of Evaluation from Arithmetic Expressions
- Creating Circles of Evaluation from Arithmetic Expressions 2
- Creating Circles of Evaluation from Arithmetic Expressions 3
- Converting Circles of Evaluation to Arithmetic Expressions
- Converting Circles of Evaluation to Arithmetic Expressions 2
- Evaluating Circles of Evaluation
- Evaluating Circles of Evaluation 2

More practice connecting Circles of Evaluation to Code

- Converting Circles of Evaluation to Code
- Converting Circles of Evaluation to Code 2

More 3-column practice connecting Arithmetic Expressions with Circles of Evaluation and Code:

- Translate Arithmetic Expressions to Circles of Evaluation & Code 3
- Translate Arithmetic Expressions to Circles of Evaluation & Code 4

More 3-column practice with negatives:

- Translate Arithmetic Expressions to Circles of Evaluation & Code 5
- Translate Arithmetic Expressions to Circles of Evaluation & Code 6

More 3-column practice with square roots:

- Translating Circles of Evaluation to Code - w/Square Roots

3-column challenge problems with brackets and exponents:

- Arithmetic Expressions to Circles of Evaluation & Code - Challenge 2
- Arithmetic Expressions to Circles of Evaluation & Code - Challenge 3
- Arithmetic Expressions to Circles of Evaluation & Code - Challenge 4

Simple Data Types

Simple Data Types

(Also available for Pyret)

Students begin to program in Pyret, learning about basic data types, and operations on those data types.

Lesson Goals	Students will be able to… • Identify examples of the following data types: Numbers, Strings, and Booleans • Write Numbers, Strings, and Booleans in the Interactions Area • Write expressions that produce values of those types
Student-facing Lesson Goals	• I can tell if a value is a Number, String or a Boolean • I can explain the different between those types
Materials	• Lesson Slides • Numbers and Strings (Page 15) • Booleans (Page 16)
Preparation	• Make sure all materials have been gathered • Computer for each student (or pair), with access to the internet

<table>
<tr><td></td><td>- Student workbook, and something to write with
- Decide how students will be grouped in pairs
- Make sure student computers can access WeScheme</td></tr>
<tr><td><b>Supplemental Resources</b></td><td></td></tr>
<tr><td><b>Key Points For The Facilitator</b></td><td>- Error messages are the computer trying to give us a clue that something is wrong. Model reacting to <b><i>error messages</i></b> with interest to demonstrate to students that the messages are a helpful tool.</td></tr>
<tr><td><b>Language Table</b></td><td><table><tr><th>Types</th><th>Functions</th><th>Values</th></tr><tr><td>Number</td><td><code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>expt</code>, <code>sqr</code>, <code>sqrt</code></td><td><code>4</code>, <code>-1.2</code>, <code>2/3</code>, <code>pi</code></td></tr></table></td></tr>
</table>

Click here to see the prior unit-based version.

Glossary

Boolean :: a type of data with two values: true and false

definitions area :: the left-most text box in the Editor where definitions for values and functions are written

editor :: software in which you can write and evaluate code

error message :: information from the computer about errors in code

interactions area :: the right-most text box in the Editor, where expressions are entered to evaluate

syntax error :: errors where the computer cannot make sense of the code (e.g. - missing commas, parentheses, unclosed strings)

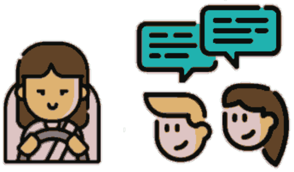
Numbers & Strings 20 minutes

Overview

Working together using a Driver/Navigator group setup, students experiment with the Editor. They explore Number and String datatypes, and how they behave in this programming language.

Launch

When programming in this class, you’ll be working together using the *Driver/Navigator* model. Each group can only have one "Driver" - their hands are on the keyboard, and their job is to manage the typing, clicking, shortcuts, etc. If you’re not a Driver, you’re a "Navigator" - your job is to tell the Driver where to go, what to type, etc. A good Driver types only what the Navigator tells them to, and a good Navigator makes sure to give clear and precise instructions.



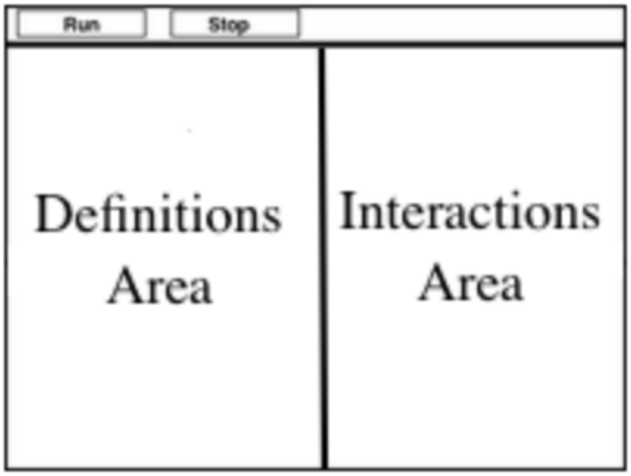
The Driver/Navigator Model

This model of pair programming is extremely useful for teasing apart the "thinking" step from the "typing" one. Students - especially those who are new to text-based programming or typing itself - can struggle to put their thoughts into the programming environment. This model allows them to focus on *communicating their ideas*, but letting the Driver focus on the coding. Likewise, the Driver has a chance to focus on syntax and programming. Finally, the requirement that ideas are translated through another person’s hands is an excellent scaffold for getting students talking about their thinking and about code.
You can read more about the Driver/Navigator model here…

Students should open WeScheme in their browser, and click "Log In". This will ask them to log in with a valid Google account (Gmail, Google Classroom, YouTube, etc.), and then show them the "My Programs" page. This page is empty - they don’t have any programs yet! Have them click "Start a New Program".

This screen is called the ***Editor***, and it looks something like the diagram you see here. There are a few buttons at the top, but most of the screen is taken up by two large boxes: the ***Definitions Area*** on the left and the ***Interactions Area*** on the right.

The ***Definitions Area*** is where programmers define values and functions that they want to keep, while the ***Interactions Area*** allows them to experiment with those values and functions. This is like writing function definitions on a blackboard, and having students use those functions to compute answers on scrap paper.



For now, we will only be writing programs in the **Interactions Area** on the right.

Investigate

Math is a language, just like English, Spanish, or any other language. Languages have **nouns** (e.g. “ball”, “tomato”, etc.) and **verbs**, which are actions we can perform on these nouns (e.g. - I can “throw a ball”). Math and programming also have **values**, like the numbers 1, 2 and 3. And, instead of verbs, they have functions, which are actions we can perform on values (e.g. - “I can square a number”).

Languages also have rules for **syntax**. In English, for example, words don’t have `!` and `?` in the middle. In math and programming numbers don’t have `&` in them.

Languages also have rules for **grammar**. *The cat sat.* is a sentence, whereas *The sat cat.* is nonsense, even though all the words are valid syntax. The order of the words matters!

Keeping the importance of **syntax** and **grammar** in mind is helpful when learning to program!.

Have students complete Numbers and Strings (Page 15). Ask them to pay special attention to the error messages!

- What did you Notice? What do you Wonder?
- Did you get any error messages? What did you learn from them? *Most of the error messages we’ve just seen were drawing our attention to* ***syntax errors****: Missing commas, unclosed strings, etc.*

Synthesize

Our programming language knows about many types of numbers, and they behave pretty much the way they do in math. Discuss what students have learned:

- Numbers and Strings evaluate to themselves.
- Our Editor is pretty smart, and can automatically switch between showing a rational number as a fraction or a decimal, just by clicking on it!
- Anything in quotes is a String, even something like `"42"`.
- Strings *must* have quotation marks on both sides.

Booleans 20 minutes

Overview

This lesson introduces students to ***Booleans***, a unique datatype with only two values: "true" and "false", and why they are useful in both the real world and the programming environment.

Launch

What’s the answer: is 3 greater than 10?

Boolean-producing expressions are yes-or-no questions and will always evaluate to either `true` (“yes”) or `false` (“no”). The ability to separate inputs into two categories is unique and quite useful!
For example, some rollercoasters with loops require passengers to be a minimum height to make sure that riders are safely held in place by the one-size-fits all harnesses. The gate keeper doesn’t care exactly how tall you are, they just check whether you are as tall as the mark on the pole. If you are, you can ride, but they don’t let people on the ride who are shorter than the mark because they can’t keep them safe. Similarly, when you log into your email, the computer asks for your password and checks whether it matches what’s on file. If the match is `true` it takes you to your messages, but, if what you enter doesn’t match, you get an error message instead.

Brainstorm other scenarios where Booleans are useful in and out of the programming environment.

Investigate

In pairs, students complete Booleans (Page 16), making predictions about what a variety of Boolean expressions will return and testing them in the editor.

Synthesize

Debrief student answers as a class.

What sets Booleans apart from other data types?

Contracts

Contracts

(Also available for Pyret)

Students learn how to apply Functions in the programming environment, encounter Image data types, and learn how to interpret the information contained in a Contract: Name, Domain and Range.

Lesson Goals	Students will be able to: • Name and explain the three parts of a Contract • Use Contracts to apply functions that produce Numbers, Strings, and Images • Demonstrate understanding of ***Domain*** and ***Range*** and how they relate to ***Functions***
Student-facing Lesson Goals	• I can make images • I can identify the Domain and Range of a function. • I can use a Contract to apply a function
Materials	• Lesson Slides • Applying Functions (Page 17) • Domain and Range Frayer model (Page 18) • Practicing Contracts: Domain & Range (Page 19)

	- Matching Expressions and Contracts (Page 20) - Using Contracts (Page 21) - Triangle Contracts (Page 23) - Radial Star (Page 24) - *Optional:* *Using Contracts (continued) (Page 22)*
Preparation	- Make sure all materials have been gathered - Computer for each student (or pair), with access to the internet - Student workbook, and something to write with - Decide how students will be grouped in pairs - All students should log into WeScheme and open the "Editor"
Key Points For The Facilitator	- Check frequently for understanding of ***data types*** and ***contracts*** during this lesson and throughout subsequent lessons. - Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location.
Supplemental Resources	

Language Table

Types	Functions	Values
Number	`+` , `-` , `*` , `/` , `expt` , `sqr` , `sqrt`	`4` , `-1.2` , `2/3` , `pi`
String	`string-repeat` , `string-contains?`	`"hello"` , `"91"`
Boolean	`<` , `<>` , `<=` , `>=` , `string<?` , `string>?` , `string=?` , `string<>?` , `string>=?`	`true` , `false`
Image	`star` , `triangle` , `square`	🔵🔺🔶

Click here to see the prior unit-based version.

Glossary

argument :: the inputs to a function; expressions for arguments follow the name of a function

contract :: a statement of the name, domain, and range of a function

contract error :: errors where the code makes sense, but uses a function with the wrong number or type of arguments

data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

domain :: the type or set of inputs that a function expects

error message :: information from the computer about errors in code

function :: a mathematical object that consumes inputs and produces an output

name :: how we refer to a function or value defined in a language (examples: +, *, star, circle)

range :: the type or set of outputs that a function produces

syntax error :: errors where the computer cannot make sense of the code (e.g. - missing commas, parentheses, unclosed strings)

variable :: a letter or symbol that stands in for a value or expression

Applying Functions 10 minutes

Overview

Students learn how to apply functions in WeScheme, reinforcing concepts from standard Algebra, and practice reading error messages to diagnose errors in code.

Launch

Students know about Numbers, Strings, and Booleans — all of which behave just like they do in math. But what about ***functions***? Students may remember functions from algebra: $f(x) = x + 4$.

- What is the name of this function? *f*
- The expression $f(2)$ applies the function f to the number 2. What will it evaluate to? *6*
- What will the expression $f(3)$ evaluate to? *7*
- The values to which we apply a function are called its ***arguments***. How many arguments does f expect? *1*

Arguments (or "inputs") are the values passed into a function. This is different from ***variables***, which are the placeholders that get *replaced* with input values! Pyret has lots of built-in functions, which we can use to write more interesting programs.
Have students log into WeScheme, open the editor, type `(sqrt 16)` into the interactions area and hit Enter.

- What is the name of this function? `sqrt`
- How many arguments does the function expect? *1*
- What type of argument does the function expect? *Number*
- Does the `sqrt` function produce a Number? String? Boolean? *Number*
- What did the expression evaluate to? *4*

Have students type `(string-length "rainbow")` into the interactions area and hit Enter:

- What is the name of this function? *string-length*
- How many arguments does `string-length` expect? *1*
- What type of argument does the function expect? *String*

- What does the expression evaluate to? *7*
- Does the `string-length` function produce a Number? String? Boolean? *Number*

Investigation

Have students complete Applying Functions (Page 17) to investigate the `triangle` function and a series of error messages. As students finish, have them try changing the expression `(triangle 50 "solid" "red")` to use `"outline"` for the second argument. Then have them try changing colors and sizes!

Synthesize

Debrief the activity with the class.

- What are the types of the arguments `triangle` was expecting? *A Number and 2 Strings*
- How does the output relate to the inputs? *The Number determines the size and the Strings determine the style and color.*
- What kind of value was produced by that expression? *An Image! New data type!*
- Which error messages did you encounter?

Contracts 15 minutes

Overview

This activity introduces the notion of ***Contracts***, which are a simple notation for keeping track of the set of all possible inputs and outputs for a function. They are also closely related to the concept of a *function machine* , which is introduced as well. *Note: Contracts are based on the same notation found in Algebra!*

Launch

When students typed `(triangle 50 "solid" "red")` into the editor, they created an example of a new ***data type***, called an *Image* .

The `triangle` function can make lots of different triangles! The size, style and color are all determined by the specific inputs provided in the code, but, if we don't provide the function with a number and two strings to define those parameters, we will get an error message instead of a triangle.

As you can imagine, there are many other functions for making images, each with a different set of arguments. For each of these functions, we need to keep track of three things:

1. **Name** — the name of the function, which we type in whenever we want to use it
2. **Domain** — the type(s) of data we give to the function
3. **Range** — the type of data the function produces

The ***Name***, ***Domain*** and ***Range*** are used to write a ***Contract***.

Where else have you heard the word "contract"? How can you connect that meaning to contracts in programming?

An actor signs a contract agreeing to perform in a film in exchange for compensation, a contractor makes an agreement with a homeowner to build or repair something in a set amount of time for compensation, or a parent agrees to pizza for dinner in exchange for the child completing their chores. Similarly, a contract in programming is an ***agreement*** *between what the function is given and what it produces.*

Contracts tell us a lot about how to use a function. In fact, we can figure out how to use functions we've never seen before, just by looking at the contract! Most of the time, error messages occur when we've accidentally broken a contract.

Contracts don't tell us *specific* inputs. They tell us the ***data type*** of input a function needs. For example, a Contract wouldn't say that addition requires "3 and 4". Addition works on more than just those two inputs! Instead, it would tells us that addition requires "two Numbers". When we *use* a Contract, we plug specific numbers or strings into the expression we are coding.

Contracts are general. Expressions are specific.

Let's take a look at the Name, Domain, and Range of the functions we've seen before:

A Sample Contracts Table

Name		Domain		Range
`; +`	`:`	`Number, Number`	`->`	`Number`
`; -`	`:`	`Number, Number`	`->`	`Number`
`; /`	`:`	`Number, Number`	`->`	`Number`
`; *`	`:`	`Number, Number`	`->`	`Number`
`; sqr`	`:`	`Number`	`->`	`Number`
`; sqrt`	`:`	`Number`	`->`	`Number`
`; <`	`:`	`Number, Number`	`->`	`Boolean`
`; >`	`:`	`Number, Number`	`->`	`Boolean`
`; <=`	`:`	`Number, Number`	`->`	`Boolean`
`; >=`	`:`	`Number, Number`	`->`	`Boolean`
`; ==`	`:`	`Number, Number`	`->`	`Boolean`
`; <>`	`:`	`Number, Number`	`->`	`Boolean`
`; string-equal?`	`:`	`String, String`	`->`	`Boolean`
`; string-contains?`	`:`	`String, String`	`->`	`Boolean`
`; string-length`	`:`	`String`	`->`	`Number`
`; triangle`	`:`	`Number, String, String`	`->`	`Image`

When the input matches what the function consumes, the function produces the output we expect.

Optional: Have students make a Domain and Range Frayer model (Page 18) and use the visual organizer to explain the concepts of Domain and Range in their own words.

Here is an example of another function. `(string-append "sun" "shine")`

Type it into the editor. What is its contract? `string-append :: String, String -> String`

Investigate

Have students complete pages Practicing Contracts: Domain & Range (Page 19) and Matching Expressions and Contracts (Page 20) to get some practice working with Contracts.

Synthesize

- What is the difference between a value like `17` and a type like `Number`?
- For each expression where a function is given inputs, how many outputs are there? *For each collection of inputs that we give a function there is exactly one output.*

Exploring Image Functions 20 minutes

Overview

This activity digs deeper into Contracts. Students explore image functions to take ownership of the concept and create an artifact they can refer back to. Making images is highly motivating, and encourages students to get better at both reading error messages and persisting in catching bugs.

Launch

Error Messages

The error messages in this environment are *designed* to be as student-friendly as possible. Encourage students to read these messages aloud to one another, and ask them what they think the error message *means*. By explicitly drawing their attention to errors, you will be setting them up to be more independent in the next activity!

Suppose we had never seen `star` before. How could we figure out how to use it, using the helpful error messages?

- Type `star` into the Interactions Area and hit "Enter". What did you get back? What does that mean? *There is something called "star", and the computer knows it's a function!*
- If it's a function, we know that it will need an open parentheses and at least one input. Have students try `(star 50)`
- What error did we get? What *hint* does it give us about how to use this function? `star` *has three elements in its Domain*
- What happens if I don't give it those things? *We won't get the star we want, we'll probably get an error!*
- If I give `star` what it needs, what do I get in return? *An Image of the star that matches the arguments*
- What is the contract for star? *star : Number String String -> Image*

- The contract for `square` also has `Number String String` as the Domain and `Image` as the Range. Does that mean the functions are the same? *No! The Domain and Range are the same, but the function name is different… and that's important because the `star` and `square` functions do something very different with those inputs!*

Investigate

- At the back of your workbook, you'll find pages with space to write down a contract and example or other notes for every function you see in this course. The first few have been completed for you. You will be adding to these contract pages and referring back to them for the remainder of this Bootstrap class!
- Take the next 10 minutes to experiment with the image functions listed in the contracts pages.
- When you've got working expressions, record the contracts and the code!

(If needed, you can print a copy of these contracts pages for your students.)

Strategies for English Language Learners

MLR 2 - Collect and Display: As students explore, walk the room and record student language relating to functions, domain, range, contracts, or what they perceive from ***error messages***. This output can be used for a concept map, which can be updated and built upon, bridging student language with disciplinary language while increasing sense-making.

Synthesize

- `square` and `star` have the same Domain *(Number, String, String)* and Range *(Image)*. Did you find any other shape functions with the same Domain and Range? *Yes!* `triangle` *and* `circle`.
- Does having the same Domain and Range mean that the functions do the same things? *No! They make very different images!*
- A lot of the Domains for shape functions are the same, but some are different. Why did some shape functions need more inputs than others?
- Was it harder to find contracts for some of the functions than others? Why?
- What error messages did you see? *Too few / too many arguments given, missing parentheses, etc.*

- How did you figure out what to do after seeing an error message? *Read the error message, think about what the computer is trying to tell us, etc.*
- Which input determined the size of the Rhombus? What did the other number determine?

Contracts Help Us Write Code 10minutes

Overview

Students are given contracts for some more interesting image functions and see how much more efficient it is to write code when starting with a contract.

Launch

You just investigated image functions by guessing and checking what the contract might be and responding to error messages until the images built. If you'd started with contracts, it would have been a lot easier!

Investigate

Have students turn to Using Contracts (Page 21), Using Contracts (continued) (Page 22) and use their editors to experiment.
Once they've discovered how to build a version of each image function that satisfies them, have them record the example code in their contracts table. See if you can figure out what aspect of the image each of the inputs specifies. It may help you to jot down some notes about your discoveries. We will be sharing our findings later.

- What kind of triangle did `triangle` build? *The* `triangle` *function draws equilateral triangles*
- Only one of the inputs was a number. What did that number tell the computer? *the size of the triangle*
- What other numbers did the computer need to already know in order to build the `triangle` function? *all equilateral triangles have three 60 degree angles and 3 equal sides*
- If we wanted to build an isosceles triangle or a right triangle, what additional information would the computer need to be given?

Have students turn to Triangle Contracts (Page 23) and use the contracts that are provided to write example expressions. If you are ready to dig into `triangle-sas`, you can also have students work through Triangle Contracts (SAS & ASA).

Sometimes it’s helpful to have a contract that tells us more information about the arguments, like what the 3 numbers in a contract stand for. This will not be a focal point of our work, but to give students a taste of it, have them turn to Radial Star (Page 24) and use the contract to help them match the images to the corresponding expressions. For more practice with detailed contracts you can have them turn to Star Polygon to work with the detailed contract for a `star-polygon`. Both of these functions can generate a wide range of interesting shapes!

Synthesize

Make sure that all students have completed the shape functions in their contracts pages with both contracts and example code so they have something to refer back to.

- How was it different to code expressions for the shape functions when you started with a contract?
- For some of you, the word `ellipse` was new. How would you describe what an ellipse looks like to someone who’d never seen one before? Why did the contract for `ellipse` require two numbers? What happened when the two numbers were the same?

How to diagnose and fix errors is a skill we will continue working on developing. Some of the errors are ***syntax errors***: a missing comma, an unclosed string, etc. All the other errors are ***contract errors***. If you see an error and you know the syntax is right, ask yourself these three questions:

- What is the function that is generating that error?
- What is the contract for that function?
- Is the function getting what it needs, according to its Domain?

Common Misconceptions

Students are *very* likely to randomly experiment, rather than to actually use the Contracts. You should plan to ask lots of direct questions to make sure students are making this connection, such as:

- How many items are in this function’s Domain?
- What is the *name* of the 1st item in this function’s Domain?
- What is the *type* of the 1st item in this function’s Domain?
- What is the *type* of the Range?

Additional Exercises:

- Matching Images to Code (Desmos)

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Function Composition

Function Composition

(Also available for Pyret)

Students encounter new image transformation functions and strengthen their understanding of Circles of Evaluation by using functions within other functions.

Lesson Goals	Students will be able to: • Use functions as building-blocks, composing them to achieve a desired affect • Diagram function composition using the Circles of Evaluation • Compose functions when programming
Student-facing Goals	• I can map a path from one number to another by composing functions • I can use Circles of Evaluation to show how functions can be composed
Materials	• Lesson Slides

- Function Cards - print and cut
- Diagramming Function Composition (Page 26)
- Function Composition — Green Star (Page 27)
- Function Composition — Your Name (Page 28)
- Function Composition — scale-xy (Page 29)
- More than one way to Compose an Image! (Page 30)
- Optional: *Function Composition Matching Activity (Desmos)*

Supplemental Resources

- Random Integer Generator
- Circles of Evaluation Review - Blank Template

Preparation

- Make sure all materials have been gathered
- Decide how students will be grouped in pairs

Key Points For The Facilitator

- Check frequently for understanding of ***data types*** and ***contracts*** during this lesson and throughout subsequent lessons.
- When students encounter errors, encourage them to check their Contracts page and show their work using Circles of Evaluation.
- Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`	🔵🔺🔶

Click here to see the prior unit-based version.

Glossary

contract :: a statement of the name, domain, and range of a function

data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

definitions area :: the left-most text box in the Editor where definitions for values and functions are written

image :: a type of data for pictures

interactions area :: the right-most text box in the Editor, where expressions are entered to evaluate

Composing Functions 10 minutes

Students are given a scaffolded activity that forces them to use the output of one function as the input to another - to *compose* them.

Launch

Divide students into groups of 3-4, and distribute a set of Function Cards to each group. Write down pairs of integers on the board, representing the "starting numbers" and "ending numbers". These integers should range from -50 to +50, but you can change the difficulty of the activity by making that span wider (more difficult) or more narrow (less difficulty). You can find a random integer generator here.

- Each group has a set of functions, each of which takes an input and produces an output. I can start with the number `4`, for example, and give it to the function `add6`. What will the output be? *10*
- I can also *compose* functions, meaning that the output of one is immediately passed into another. For example, I could compose `add6` and `double`, so the `10` gets passed into the next function, and doubled to produce `20`. What would happen if I composed `add6` with `double` *and* with `half`? *10*
- For each of the starting numbers on the board, your job is to figure out which functions to compose in order to get to the end.
- You will need to use some functions more than once, and that's ok!

Investigate

Give students time to experiment with this. You can make the activity more challenging by asking them to find the *shortest path* from start to end, using the smallest number of compositions.

Synthesize

If two groups come up with different compositions that achieve the same end result, have them share their ideas!

Diagramming Function Composition

Overview

The Circles of Evaluation are extended to provide a visual-spatial metaphor for function composition, in addition to Order of Operations.

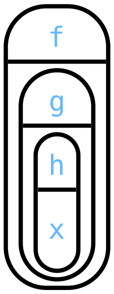
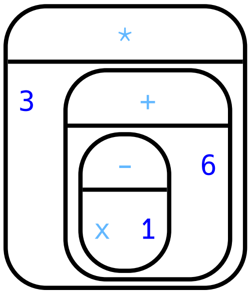
Launch

Three of the function cards we just used were for the functions `f`, `g` and `h`:

- `f` multiplied its input by 3
- `g` added six to its input
- `h` subtracted one from its input

We can compose those function in any order. If we composed them as `f(g(h(x)))` and evaluated them for `x = 4` what would happen?

We can diagram the function composition using Circles of Evaluation (see first column, below). In the second column, we've replaced the function names in each Circle of Evaluation with *what each function does*:

Function Composition	Order of Operations
	

The circles show us that in order to evaluate $f(g(h(4)))$

- First we would have to evaluate $h(4)$, subtracting `1` from `4` to get `3`
- Then we would evaluate $g(3)$, adding `6` to `3` to get `9`
- Then we would evaluate $f(27)$, tripling `9` to get `27`

Investigate

Have students turn to Diagramming Function Composition (Page 26) to practice writing, translating and evaluating Circles of Evaluation of composed functions.

Synthesize

Do $f(g(h(x)))$ and $g(h(f(x)))$ evaluate to the same thing? *No!*
Why not? *order matters!*

Composing Functions in Code 20 minutes

Overview

The Circles of Evaluation are extended to functions that do more than compute values.

Launch

The contracts page in your workbook is just like the Function Cards from this activity. Your job as a programmer is to figure out how to compose those functions to get where you want to go, in the most clever or elegant way possible.

Investigate

Have students log into WeScheme open a new program and save it as Function Composition.
Have students open to Function Composition — Green Star (Page 27), in which they will be drawing circles of evaluation to help them write expressions to compose a series of images.

- Make sure students are using the ***Definitions area*** (left side) for code they want to keep and are using the ***Interactions area*** (right side) to test code or try out new ideas.
- When students are finished, check their work, and ask them to change the color of all of the stars to “gold” or another color of your choosing.

Then have students open to Function Composition — Your Name (Page 28) in which they will create a text ***image*** of their name and experiment with other functions.

Strategies for Facilitation

While students are exploring, be available for support but encourage student discussion to solve problems. Many student questions can be addressed with these responses: *Did you try drawing the Circle of Evaluation first? Did you check the contract? Have you pressed the Run button to save your Definitions changes?*
Encourage students to practice writing comments in the code to describe what is being produced, using `;` at the beginning of the line.

If you have time, you can also have students work with Function Composition — scale-xy (Page 29) and/or Function Composition Matching Activity (Desmos)

Synthesize

- What do all of these functions have in common? *They all produce images, they all change some element of the original image*
- Does using one of these functions change the original image? *No, it creates a whole new image*
- What does the number in `scale` represent? *The scale factor by which the image should grow or shrink*
- What does the number in `rotate` represent? *The rotation angle, measured counterclockwise*
- The Domain and Range for `flip-horizontal` is Image -> Image. Why can I use the output of the `text` function as an *input* for `flip-horizontal`? *Because the `text` function produces an Image, which is then used as the input for `flip-horizontal`.*

Strategies for English Language Learners

MLR 1 - Stronger and Clearer Each Time: As an alternative, display the discussion questions during the last 5 minutes of the Explore and ask students to discuss the questions with their partner, asking each other for explanation and details and coming up with the clearest, most precise answer they can. Student pairs can then share with another pair and compare their responses before moving into a full class discussion.

Fun with Images!

Now that students have learned how to use all of these image-composing functions, you may want to give them a chance to create a design of their own, tasking them with using at least 4 functions to create an image of their choosing.

Our Flags lesson also dives deeper into image composition.

Composing Multiple Ways Optional

Overview

Students identify multiple expressions that will create the same image, and think about the merits of one expression over another.

Launch

As is often true with solving math problems, there is more than one way to get the same composed image.
Suppose I wrote the code: `(scale 3 (star 50 "solid" "red"))`.
What's another line of code I could write that would produce the exact same image?

```
(star 150 "solid" "red")
```

Investigate

Students complete More than one way to Compose an Image! (Page 30).

Synthesize

There is a special function in WeScheme that let's us test whether or not two images are equal.

```
image=? :: Image, Image -> Boolean
```

Use it to test whether all of the expressions you wrote successfully build the same images.

- Could we have written more expressions to create the same images?
- Are all of the ways to write the code equally efficient?

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Defining Values

Defining Values

(Also available for Pyret)

Students learn how to define names for computed values. These names can be used repeatedly in different situations, just like variables in math.

Lesson Goals	Students will be able to: • Define variables of various types • Simplify a complex expression by replacing repeated parts with defined names, • Explain why ***variable***s are useful in math and programming
Student-facing Goals	• I can define names for ***values***, generated by computed expressions • I can clean up complex code by defining repeated expressions • I can explain why variables are important
Materials	• Lesson Slides • Defining Values - Explore (Page 32) • Defining Values - Chinese Flag (Page 33) • Why Define Values? (Page 34) • Writing Code using Defined Values (Page 36) • Defining Values Starter File (Wescheme) • Chinese flag Starter File (Wescheme) • *Optional: Matching Code to Images using overlay & put-image (Desmos)*
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs

Key Points For The Facilitator

- Learning how to define values is a big milestone! It will be used consistently throughout other lessons, so be sure to give students plenty of time to practice this new skill.
- Check frequently for understanding of ***data types*** and ***contract***s during this lesson and throughout subsequent lessons.
- Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version.

Glossary

contract :: a statement of the name, domain, and range of a function

data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

definitions area :: the left-most text box in the Editor where definitions for values and functions are written

value :: a specific piece of data, like 5 or "hello"

variable :: a letter or symbol that stands in for a value or expression

What’s in Common? 30 minutes

Overview

This activity introduces the problem with duplicate code, leveraging **Mathematical Practice 7 - Identify and Make Use of Structure** . Students identify a common structure in a series of expressions, and discover how to bind that expression to a name that can be re-used.

Launch

Take a look at the expressions below:

```
(star 50 "solid" "green")
(scale 3 (star 50 "solid" "green"))
(scale .5 (star 50 "solid" "green"))
(rotate 45 (star 50 "solid" "green"))
(rotate 45 (scale 3 (star 50 "solid" "green")))
```

- **What code do they all have in common?** `(star 50 "solid" "green")`
- **What would happen if you were asked to change the color of all the stars to gold?** *We’d have to change it everywhere it appeared.*

Duplicate code is almost always bad!

There are lots of potential problems with duplicate code:

- **Readability:** The more code there is, the harder it can be to read.
- **Performance:** Why re-evaluate the same code a dozen times, when we can evaluate it *once* and use the result as many times as we need?
- **Maintainability:** Suppose we needed to change the size of the stars in the examples above. We would have to make sure every line is changed, which leaves a lot of room for error.

Since we’re using that star over and over again, wouldn’t it be nice if we could define a "nickname" for that code, and then use the nickname over and over in place of the expression? And then, if we wanted to change something about all of the stars, we would only have to make the change once, in the definition.
You already know how to do this in math:

$x = 4$ *defines* the nickname x to be the value 4.

WeScheme uses the word "define" to make this even clearer!

We can type `(define x (4))` to define `x` to be the value 4.

Once we've defined `x` to be the value 4 and clicked "run", anytime we use `x`, the computer will remember that it is *defined* as 4. We can *use* that definition to get an answer. For example, $x + 2$ will evaluate to 6.

Of course, the whole point of defining a value is so that it sticks around and can be used later! That's why programmers put their definitions on the *left-hand side*, known as the ***Definitions Area***.

Investigate

- Open the Defining Values Starter File (Wescheme) and complete the notice and wonder exercise on Defining Values - Explore (Page 32) with your partner.
- Complete the remaining questions and add some defintions of your own in the definitions area. Be sure to click **Run** again before you try testing them out.

Synthesize

- **What data types can we define values for?** *All of them - Number, String, Image*
- What values did you decide to define? When might they be useful?

Support for English Language Learners

MLR 8 - Discussion Supports: As students discuss, rephrase responses as questions and encourage precision in the words being used to reinforce the meanings behind some of the programming-specific language, such as "define" and "value".

Using Defined Values

Overview

Now that we know *how* to define values, we've got two more things to consider:

- When it would be *useful* to define them
- How to *use* them once we have

Launch

Have students open to Defining Values - Chinese Flag (Page 33). It will direct them to open the Chinese flag Starter File (Wescheme)
once they complete the first half of the questions on the page.

Investigate

- Have students open the editor and type `(radial-star 30 20 50 "solid" "yellow")` into the interactions area and click run.
- Have students work in the Definitions area to define a value `sun` to be the image `(radial-star 30 20 50 "solid" "yellow")`.
- Turn to Why Define Values? (Page 34). The first row of the table has been completed for you. Could I get a volunteer to explain what they see happening in that first row?
- Have students complete the page and test their code in the editor.

Synthesize

- **Why is defining values useful?** *Lets the programmer reuse code, saves time, lets the programmer make changes easily, allows us to more easily use elements inside other functions*

Optional: Fun with Flags!

Now that students have learned how to define values, you may want to give them an opportunity to practice. Our Flags lesson dives deeper into image composition and includes some starter files specifically focused on cleaning up code by defining values.

Additional Exercises:

- Writing Code using Defined Values (Page 36)
- Matching Code to Images using overlay & put-image (Desmos)

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

Making Flags

Making Flags

(Also available for Pyret)

Students compose the image functions they've learned, applying their knowledge of coordinates to position differently-shaped and transformed images to create flags of varying complexity.

Lesson Goals	Students will be able to: • Put one image on top of another, using the `put-image` function. • Decompose complex images into parts. • Combine and manipulate images to create more complex images.
Student-facing Goals	• I can put one image on top of another. • I can make complex images like flags.
Materials	• Lesson Slides

- Flags Starter File
- Flags of Netherlands, Ireland, Mauritius Starter File
- Estimating Coordinates (Page 37)
- Decomposing Flags (Page 38)
- Scaling Flag Ratios (Desmos)
- Matching Code to Images using overlay and put-image (Desmos)
- Optional: *Flags of the World*
- Optional: the Alaska Flag Starter Code (WeScheme)

Supplemental Resources

- Flags of the World
- Flag Wizard - Searchable index of the world's flags, based on geometric properties and complexity.

Preparation

- Make sure all materials have been gathered.
- Decide how students will be grouped in pairs.
- For some students, the use of graph paper for this activity will make a big difference!

Key Points For The Facilitator

- The `put-image` function treats the second image as the first quadrant of a cartesian plane. Having students sketch their flag on a sheet of graph paper drives this point home, and makes the programming easier.
- This is an excellent opportunity to introduce students to *indenting code*. The difference between poorly-indented flag code and well-indented flag code is extremely noticable.

Language Table

Types	Functions	Values
Number	`+` , `-` , `*` , `/` , `expt` , `sqr` , `sqrt`	`4` , `-1.2` , `2/3` , `pi`
String	`string-length` , `string-repeat` , `string-contains?`	`"hello"` , `"91"`
Boolean	`<` , `<>` , `<=` , `>=` , `string<?` , `string>?` , `string=?` , `string<>?` , `string>=?`	`true` , `false`

	Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Glossary

contract :: a statement of the name, domain, and range of a function

Putting Images Together 25 minutes

Overview

Students learn about the `put-image` function.

Launch

As you've already seen, `overlay` sticks two images together, so that the *center* of the first image is placed exactly on top of the *center* of the second image. But what if we want to put the dot somewhere besides the center?

The `put-image` function works like `overlay`, but instead of placing the centers of each image on top of one another, it *translates* the center of the top image by some distance in the x- and y-direction.

> Think of the background image as a sheet of graph paper with the origin (0,0) in the bottom left corner.
> The numbers in `put-image` specify a point on that graph paper, with the center of the top image being placed there.

- Distribute [Estimating Coordinates (Page 37)] or project the flag images from the peardeck.
- The code beneath each image is missing the x and y coordinates to place the dot.
- Estimate the x- and y-coordinate of the dot's location for each image and complete the code!

Investigate

Have students open the [Flags Starter File], and click Run.

There are some special lines in this file called **comments**. The programmer who wrote this code included a series of messages - called comments - that will never be read by the computer.

> Professional programmers use comments *all the time*.

Professional programmers work with teams who need to be able to follow each other's thinking in order to collaborate efficiently. Comments are a way for programmers to leave notes for one another, and even for a single programmer to keep track of their thinking when *they* come back to their code another day.

- Each language has its own symbol for commenting. In WeScheme, we use the semi colon (`;`) to tell the computer to ignore what we're typing. What color does the code turn when it has a semicolon in front of it? *Purple!*
- Type `japan-flag` into the Interactions Area. What do you get back?
- Type `japan` into the Interactions Area and compare the image to `japan-flag`.
- `japan` is composed using `dot` and `background`. Type each of those variables into the Interactions Area. What do you get back?
- Take a look at the `japan` code and fix it so that it matches the `japan-flag` image.
- What is the ***Contract*** for `put-image`? (Write it in your Contracts page!)

Have students open the Flags of Netherlands, Ireland, Mauritius Starter File , and click Run.

- Look at the code for the flag of the Netherlands.
- How big is the flag?
- What was the programmer thinking when she coded the height of the red stripe as `(/ 200 3)`?
- The center of the blue stripe is placed at (`150`, `(/ 200 6)`).
- How did the programmer know to use 150 as the x-coordinate?
- What was the programmer thinking when she coded the y-coordinate as `(/ 200 6)`?
- Explain the thinking behind coding the redstripe's y-coordinate as `(* 5 (/ 200 6))`.

Once you've discussed the code for the flag of the Netherlands with your class, have them keep working with this starter file to write code for the flags of Ireland and Mauritius. Some students will make it through the challenges. Some students may only complete one flag. All of them will be synthesizing how to apply put-image and locate images on the coordinate grid.

Synthesize

Could we completely replace `overlay` with `put-image`? Why or why not?

Going Deeper

If you have time, we have lots of additional starter files to push student thinking linked in the additional exercises at the end of this lesson and now would be the time to dive into them!

Making Flags 25 minutes

Overview

Students focus on decomposing complex images into simple ones, and using `put-image` to combine them.

Launch

Let’s dig into the process for how the flags we’ve seen so far were made:

1) Decompose the Image

We observe that the Japanese flag is made up of two simpler images: a black, outline rectangle and a red dot.

2) Define those parts

We define `dot` and `background`. Once we’ve defined those images, we test them out in the Interactions Area to make sure they look right!

3) Find the Coordinates

For each image, calculate what the x- and y-coordinates of the center should be. *TIP: this is a lot easier if you have a sheet of graph paper handy!*

4) Build the Image & Check that it Matches the Target Image

We stack the parts on top of the bottom image using the coordinates we found. *TIP: don’t cram all the code into one line! If you break it up into new lines (for example, hitting "Return" before the x-coordinate and after the y-coordinate), you’ll notice that the code forms a "staircase" pattern.* Be sure to compare the image you get with the target image!

If you have time, these matching activities will support student thinking.

- Scaling Flag Ratios (Desmos)
- Matching Code to Images using overlay and put-image (Desmos)

Investigate

Paper Flag Models to Code

In this next exercise, students will be decomposing the image of a flag. For a more tactile experience, you could have students construct images of the flags they choose using construction paper. This should happen between the step where they describe the shapes needed to compose the flags image and write the code to build the image. The act of physically building the flag from layers of paper makes the layering of the coded images visible and helps students to remember that white space is not just "blank".

- Turn to Decomposing Flags (Page 38), and choose ONE flag to focus on. On the blank lines below, describe the parts that make up that flag.
- Once you’re done, return to the Flags Starter File and define those parts.
- Then, compose those parts using `put-image`, and make your flag!

Around the World in your Classroom

The opportunity to focus on a flag of their choosing is a big motivator for students. Coding flags also presents a rare opportunity for students to share a piece of their identity in math class. If you have more time to devote, we highly encourage you to give students the opportunity to create the image of a flag they connect with in some way. Students might choose the flags of countries related to their heritage, a place they’ve visited, a country they’re interested in, or a group they identify with or support. Encourage students to choose an appropriately challenging flag. The teacher may introduce parameters if necessary, such as “Flags need at least 5 different shapes”. This is intended to be a summative project, so encourage students to demonstrate what they’ve learned. Once students have coded their flags, host a tour of the flags of the world in your classroom!

Be mindful of the fact that not every student will know their family's geographical history, and that immigration can be a sensitive topic for some students. For this reason, it is important that students have the option to choose a flag based on interest instead of just family history. Be prepared that students might choose flags representing groups other than countries. Indigenous students might choose a flag that represents their indigenous nation or the American Indian Movement. Students might also choose to focus on the pride flag (representing solidarity amongst members of the LGBTQ community) or an ethnic flag (representing solidarity amongst members of different ethnic groups, such as the Hispanic flag). There are also some flags that represent political or activist groups. It is important for the teacher to be open to different beliefs and ideologies, but it is also up to the teacher's discretion as to whether or not a flag is appropriate for use in this project. Flags associated with hate groups, or any institution that denies the dignity of other students, are not appropriate.

Flags of the World and Flag Wizard may be useful to students looking for ideas. Flags are listed with their width:length ratios in the opposite order of how we define the sides of a flag in code. e.g. 2:3 could be scaled up to a 300 x 200 flag or 8:11 could be scaled up to 330 by 240.

Ratio and Proportion

Have students define the `WIDTH` and `HEIGHT` of their flags as values, and then *replace the numbers in each flag* with expressions relative to width and height. For example, if the `dot` in the Japanese flag is at (150, 100), those numbers would be replaced with `(/ WIDTH 2)` and `(/ HEIGHT 2)`.

Synthesize

Which flags were the easiest to make? The hardest?

Why is it useful to define each part of the flag first, before stitching the images together?

Additional Exercises

- For a quick dive into why it's more efficient to define shapes before building the image, open the Alaska Flag Starter Code. (Wescheme)
- For practice scaling imported graphics, open the Flag of Lebanon Starter Code. (Wescheme)
- For practice with composing more complex images, fix this Code for the Puerto Rican flag. (Wescheme)
- If you've already studied Pythagorean Theorem and are ready to apply it, open the Flag of Trinidad and Tobago Starter Code. (Wescheme)

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Making Game Images

Making Game Images

(Also available for Pyret)

Students practice using a new function alongside previously-learned functions to choose images for their game.

Lesson Goals	Students will be able to: • Apply previous knowledge of ***functions*** to new situations • Use reasoning skills to select appropriate functions and combine their effects

Student-Facing Lesson Goals	• I can use different functions to transform ***images***. • I can write definitions for my transformed images.
Materials	• Lesson Slides • Linking Images Guide - PDF • Blank Game Starter File (Wescheme) • *Optional: quick guide to saving images to drive*
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Supplemental Resources	
Key Points for the Facilitator	• Discuss copyright and fair use guidelines with your students. • Instructional time may vary based on students' experience with using Google Image Search. • Check beforehand for any issues the school Internet security blocker might cause with searching for images. • There are two ways of importing images: linking directly to the image on the web or downloading the image to Google Drive and then using the "Insert" button. See the "Linking Images Guide" below for more information on linking directly. • Encourage students to focus on finding and scaling each image as needed before moving on to the next one.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
		`true`, `false`

	Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	
	Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version.

Glossary

define :: to associate a descriptive name with a value

function :: a mathematical object that consumes inputs and produces an output

image :: a type of data for pictures

interactions area :: the right-most text box in the Editor, where expressions are entered to evaluate

The Game Starter File 15 minutes

Overview

This activity is primarily about *review and reading comprehension* , in which students open a large and unfamiliar file and must make sense of it using what they've seen before.

Launch

Students should have their workbook, pencil, and be logged into WeScheme and have their completed "Game Design" worksheet.

By now you've learned about defining values, composing functions, and reading contracts. Taken together, that's a lot of code you're now able to understand! It's time to flex your reading skills, and look at the file you'll be working with to build your video game.

This file has code you haven't seen before! And that's ok! For now, see what parts you recognize, and make sure you understand them.

Investigate

With their partner, students should load the Blank Game Starter File (Wescheme).

Notice and Wonder

As students investigate the Game Starter File file with their partner, ask students to record what they Notice, and then what they Wonder.

Synthesize

- **What familiar things did you see in the Game Starter File file?**
- **What were some unfamiliar things? Any idea what they might do?** *Answers vary: new functions, comments, images*
- **What data type is `GAME-TITLE`? What data type is `BACKGROUND`?** `GAME-TITLE` *is a String,* `BACKGROUND` *is an Image*
- **What does `SCREENSHOT` return in the *Interactions area*?** *An image of the* `BACKGROUND`, `PLAYER`, `TARGET`, *and* `DANGER` *all together*

- **Did anyone try pressing "Run"? What happens when you press "Run"?** *Allow students to discuss what they see and what connections they see with the code*
- **What do you think `image-url` does?**

What is SCREENSHOT?

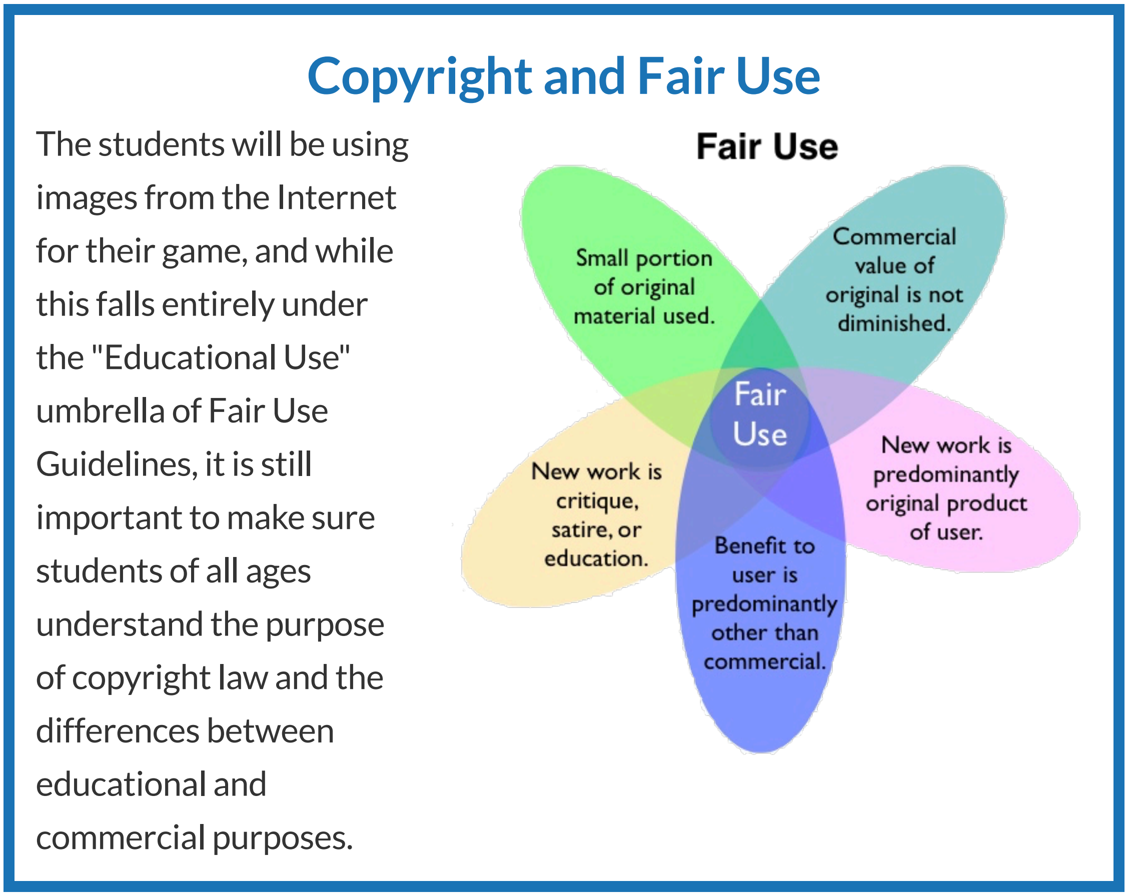
The Game Starter File defines several image values, such as `BACKGROUND`, `PLAYER`, etc. These definitions are using the running game, which appears when you click "Run". `SCREENSHOT` is defined as a fixed composition of the game images, placing each of them on top of the background at various (x,y) coordinates. It is used to give students a chance to see their characters onscreen before they've gotten them moving, and to give teachers an opportunity to reviw coordinates. It is *not* in any way connected to the running game, so changes made to `SCREENSHOT` will not impact the game that appears when clicking "Run".

Finding Your Game Images flexible

Overview

This activity is all about finding the right images for students' games. Since the internet never has *exactly* the right image, students' need to get their games **just right** motivates them to confront the need for dilation, rotation, and reflection of the images they find. This, in turn feeds back into their understanding of Contracts and Function Composition.

Launch



Copyright and Fair Use

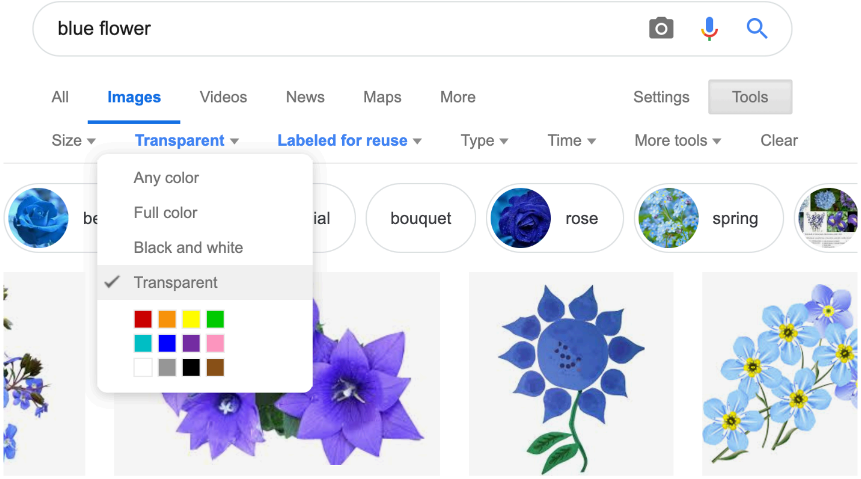
The students will be using images from the Internet for their game, and while this falls entirely under the "Educational Use" umbrella of Fair Use Guidelines, it is still important to make sure students of all ages understand the purpose of copyright law and the differences between educational and commercial purposes.

Fair Use
Small portion of original material used.
Commercial value of original is not diminished.
Fair Use
New work is critique, satire, or education.
New work is predominantly original product of user.
Benefit to user is predominantly other than commercial.

Guide the students through finding an image, saving it to their Drive, importing it into their program, and defining the image value as `PLAYER`. *Students will change this image later on their own, this is just for teaching purposes.*

How to find and save images to Google Drive....

In your favorite search engine (we recommend DuckDuckGo), search for an image and then click "Images". Click "All Types" and select "Transparent" (In Google Image Search, it’s under "Color -> Transparent"). This will filter and display images that have a transparent background, appearing as a light white/grey checkerboard pattern behind the character.



Once an image has been selected, click it to expand and save the image to Google Drive. For file management, students may want to create a folder to store their game images.

- If using a Chromebook, this is done by right-clicking and selecting "Google Drive" on the left for the save location.
- On a PC or Mac follow this, quick guide to saving images to drive.

Once the image is saved to Google Drive, it can be brought into the program by using the "Images" button. This will automatically bring in the image using the `bitmap-url` function, and students can run the code to see the image.

Investigate

What happens if the image we find needs to be made bigger or smaller? What if it needs to be rotated, or flipped?
Students can define the image as a value and make changes to it with the image manipulation functions `scale`, `rotate`, `flip-horizontal`, and `flip-vertical`.

Strategies for English Language Learners

MLR 8 - Discussion Supports: As students discuss, rephrase responses as questions and encourage precision in the words being used to reinforce the meanings behind some of the functions, such as `scale` and `flip-horizontal`.

With their partner, students search the Internet for images to use in their game. They will need 4 images, one for each visual element of their game:

- `BACKGROUND`
- `PLAYER`
- `DANGER`
- `TARGET`

Students should:

- Save the chosen images to their Drive
- Bring them into the programming environment
- ***Define*** the images as values
- Plan out how to resize and reorient them in their game
- Make sure the final version of each image is defined as either `BACKGROUND`, `TARGET`, `DANGER`, or `PLAYER`

When finished, students should be able to type `SCREENSHOT` in the interactions window and see all four of their images appropriately sized and oriented.

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

Defining Functions

(Also available for Pyret)

Students discover functions as an abstraction over a programming pattern, and are introduced to a structured approach to building them called the Design Recipe.

Lesson Goals	Students will be able to: • identify patterns where a function would be useful • explain the difference between defined *values* and *functions* • match examples, contracts, and definitions for the same function
Student-Facing Lesson Goals	• I can explain why a function is useful • I can connect contracts, examples, and definitions for a function
Materials	• Lesson Slides • Matching Examples and Definitions (Math) (Page 41) • Matching Examples and Function Definitions (Page 42) • Matching Examples and Contracts (Page 43) • Contracts, Examples & Definitions (Page 44) • the gt starter file (Wescheme) • Optional: *Contracts, Examples & Definitions - 2* • Optional: *Matching Examples & Function Definitions (Desmos)* • Optional: *Matching Examples & Function Definitions (Desmos)*
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Key Points for the Facilitator	• This lesson represents a *big* shift in thinking. After some practice, students will not be limited to pre-existing functions!

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version

Glossary

example :: shows the use of a function on specific inputs and the computation the function should perform on those inputs

function :: a mathematical object that consumes inputs and produces an output

function definition :: code that names a function, lists its variables, and states the expression to compute when the function is used

syntax :: the set of rules that defines a language, whether it be spoken, written, or programmed.

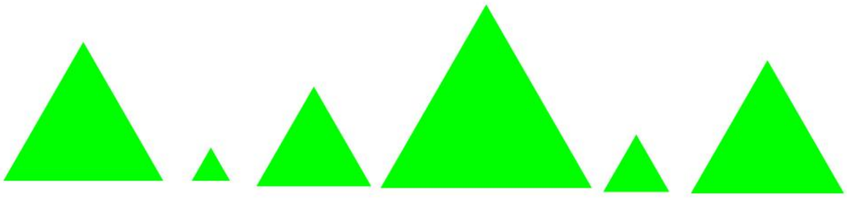
There’s Got to Be a Better Way! 15 minutes

Overview

Students have already searched for structure in a list of expressions in order to define values.
In this lesson, students will build their flexibiltiy of thinking by engaging with multiple representations. Students will search for structures that are *dynamic* , meaning they change in a predictable way. This is the foundation for defining functions.

Launch

Students should have their workbook, pencil, and be logged into WeScheme on their computer.



This is a fun lesson to make silly! Dramatically confess to your students, "I LOVE green triangles!" Challenge them to use the Definitions Area to code as many unique, solid, green triangles as they can in 2 minutes.
Walk around the room and give positive feedback on the green triangles. When the time is up, ask for some examples of green triangles that they wrote and copy them to the board. Be specific and attend to precision with the ***syntax*** such that students can visually spot the pattern between the different lines of code.

For example:

```
(triangle 30 "solid" "green")
(triangle 12 "solid" "green")
(triangle 500 "solid" "green")
```

- **Is there a pattern?** *Yes, the code mostly stayed the same with one change each time.*
- **What stayed the same?** *The function name* `triangle`*, "solid", "green".*
- **What changed?** *The size of the* `triangle`*, or the Number input.*
- **How many of you typed out the code from scratch each time? How many triangles were you able to code in a minute?** *Write this down so that you can compare to it later!!!*
- **Did you know that there is a keyboard shortcut for making the previous line of code reappear in the interacions area?** *alt/option up-arrow*

Investigate

Our programming language allows us to define **values** . This lets us create "shortcuts" to reuse the same code over and over.
For example: `(define PRIZE-STAR (star 65 "solid" "pink"))`
This code will let us write `PRIZE-STAR` wherever we want that same solid, pink star - without having write all the code again and again.
But to make a shortcut that *changes* such as creating solid, green triangles of a changing size, we need to define a *function* .
Suppose we want to define a shortcut ***function*** called `gt`. When we give it a number, it makes a solid green triangle of whatever size we give it.

Select a student to *act out* `gt`. Make it clear to the class that their Name is "gt", they expect a Number, and they will produce an Image. Act out some examples before having the class add their own and record them on the board:

- You say: **gt 20!** The student responds: *`(triangle 20 "solid" "green")`!*
- You say: **gt 200!** The student responds: *`(triangle 200 "solid" "green")`!*
- You say: **gt 99!** The student responds: *`(triangle 99 "solid" "green")`!*

Synthesize

Thank your volunteer. Assuming they did a wonderful job, ask them:

- How did you get to be so speedy at building green triangles? You seemed so confident! *Ideally they'll tell you that they had good instructions and that it was easy to follow the pattern*

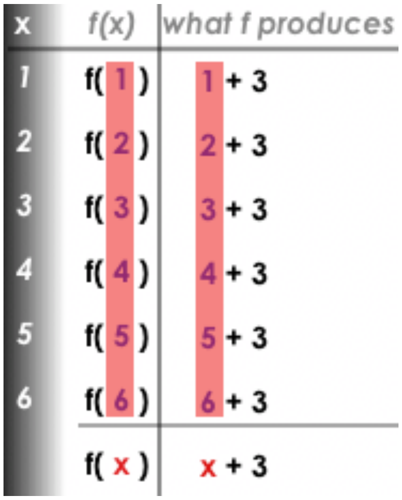
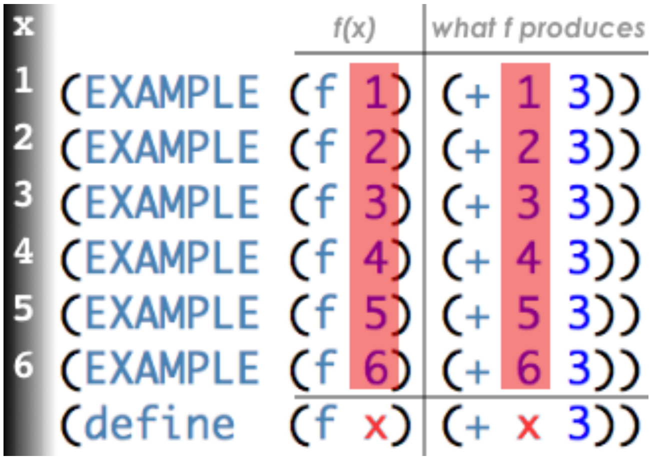
Just as we were able to give our volunteer instructions that let them take in `gt 20` and give us back `(triangle 20 "solid" "green")`, we can define any function we'd like in the **Definitions Area**.

Examples and Definitions

Launch

We need to program the computer to be as smart as our volunteer. But how do we do that? We already know how to do this in math!

- Draw the table on the left below on the board.
- We recommend starting by showing it without the equation at the bottom and talking students through the process of highlighting the variable & defining the function.
- Once you have crowd-sourced the equation from the math side, show students how the same process of writing examples and defining the function would work in Pyret syntax.

Math		WeScheme
 x | f(x) | what f produces 1 | f(1) | 1 + 3 2 | f(2) | 2 + 3 3 | f(3) | 3 + 3 4 | f(4) | 4 + 3 5 | f(5) | 5 + 3 6 | f(6) | 6 + 3 | f(x) | x + 3	→	 x | f(x) | what f produces 1 (EXAMPLE (f 1) (+ 1 3)) 2 (EXAMPLE (f 2) (+ 2 3)) 3 (EXAMPLE (f 3) (+ 3 3)) 4 (EXAMPLE (f 4) (+ 4 3)) 5 (EXAMPLE (f 5) (+ 5 3)) 6 (EXAMPLE (f 6) (+ 6 3)) (define (f x) (+ x 3))

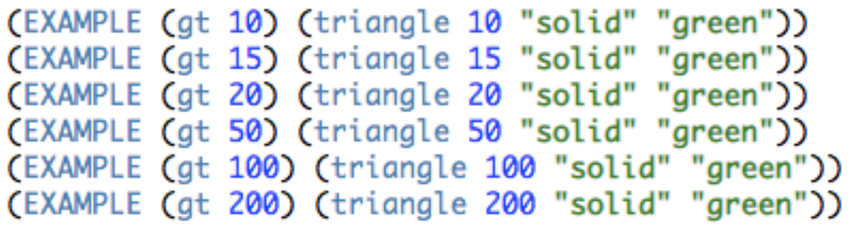
Investigate

Have students turn to Matching Examples and Definitions (Math) (Page 41).

- Start by looking at each table and highlighting what is changing from the first row to the following rows.
- Then, match each table to the function that defines it.

You may also want to have students complete Matching Examples & Function Definitions (Desmos)

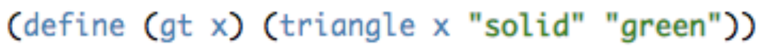
Now that we've seen how this works in math, let's go back to `gt`.



```
(EXAMPLE (gt 10) (triangle 10 "solid" "green"))
(EXAMPLE (gt 15) (triangle 15 "solid" "green"))
(EXAMPLE (gt 20) (triangle 20 "solid" "green"))
(EXAMPLE (gt 50) (triangle 50 "solid" "green"))
(EXAMPLE (gt 100) (triangle 100 "solid" "green"))
(EXAMPLE (gt 200) (triangle 200 "solid" "green"))
```

In the case of `gt`, the domain was a number and that number stood for the `size` of the triangle we wanted to make. Whatever number we gave `gt` for the size of the triangle is the number our volunteer inserted into the `triangle` function. Everything else stayed the same no matter what! We need to define `gt` in terms of the variable `size`, instead of in terms of a specific number.

Turn to Matching Examples and Function Definitions (Page 42) and look at the definition of `gt` in the first row of the table.



```
(define (gt x) (triangle x "solid" "green"))
```

Using `gt` as a model, match the mystery function examples to their corresponding definitions.

You may also want to have students complete Matching Examples & Function Definitions (Desmos).

Connecting to Best Practices

- Writing the examples is like "showing your work" in math class.
- Have students circle what is changing and label it with a proper variable name. The name of the variable should reflect what it represents, such as `size`.
- Writing examples and identifying the variables lays the groundwork for writing the function, which is especially important as the functions get more complex. Don’t skip this step!

Synthesize

- What strategies did you use to match the ***examples*** with the ***function definitions***?
- Why is defining functions useful to us as programmers?

Examples and Contracts

Launch

- What is the contract for `triangle`?

  ```
  triangle :: Number, String, String -> Image
  ```

- What is the contract for `gt`?

  ```
  gt :: Number -> Image
  ```

- Why might someone think the domain for `gt` contains a Number and two Strings? *The function* `gt` *only needs one Number input because that's the only part that's changing. The function* `gt` *makes use of* `triangle`*, whose Domain is Number String String, but* `gt` *already knows what those strings should be.*

Investigate

Have students turn to Matching Examples and Contracts (Page 43).
Confirm that everyone is on the same page before moving on. You may want to have students turn to a partner, compare their findings, and discuss their thinking about anything they didn't agree on at first.
Have students open the gt starter file (Wescheme).

- Click **Run** and evaluate `(gt 10)` in the Interactions Area.
- What did you get back? *a little green triangle!*
- Take one minute and see how many different green triangles you can make using the `gt` function.
- Try changing one of the examples to be incorrect and click run again. What happens? *The editor lets us know that the function doesn't match the examples so that we can fix our mistake!*

Have students turn to Contracts, Examples & Definitions (Page 44)

> On the top half of the page you will see the contract, examples, and function defintion for `gt`. Using `gt` as a model, complete the contract, examples and function defintion for `bc`. Then type the Contract, Examples and Definition into the Definitions Area, click "Run", and make sure all of the examples pass!

If you have time, have students complete

- Contracts, Examples & Definitions - 2
- Bug Hunting in WeScheme (Wescheme)

Synthesize

- **Functions can consume values besides Numbers. What other datatypes did you see being consumed by these functions?**
- Thumbs up? Thumbs to the side? or Thumbs down? How confident do you feel that you could write the contract, examples and function definition on your own if you were given a word problem about another shape function?

Additional Exercises:

- Matching Examples & Contracts (Desmos)

Solving Word Problems

Solving Word Problems

(Also available for Pyret)

Students discover functions as an abstraction over an arithmetic pattern, applying the Design Recipe to traditional word problems.

Lesson Goals	Students will be able to: • Understand how to use the Design Recipe to break down word problems. • Create a strong purpose statement that details in their own words what the function should do.
Student-Facing Lesson Goals	• I can write my own function in WeScheme . • I can use the ***Design Recipe*** to break down word problems when writing a ***function***.
Materials	• Lesson slides

- Creating Contracts From Examples (Page 47)
- Writing Examples from Purpose Statements (Page 48)
- Word Problem: rocket-height (Page 49)
- rocket-height Starter File (Wescheme)
- *Optional: Writing Examples from Purpose Statements 2*
- *Optional: Do Examples Have the Same Contracts?*
- *Optional: Do Examples Have the Same Contracts? (2)*

Preparation

- Make sure all materials have been gathered
- Decide how students will be grouped in pairs

Supplemental Resources

Design Recipe Practice - Desmos Activity

Key Points for the Facilitator

- The **purpose statement** is a comment in the code - something the computer doesn't read. It is important for readability of their code - there may be other people looking at their code and using their functions!
- Remind students that the domain and range of a function must be one or more of the ***data types*** (Number, String, Image, Boolean, etc.) they've learned so far.
- If students struggle with creating the examples, use the Circle of Evaluation mapping activity or use role-playing to help students build up their understanding around the concept.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
		● ▲ ◆

	Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	

Click here to see the prior unit-based version

Glossary

contract :: a statement of the name, domain, and range of a function

data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

design recipe :: a sequence of steps that helps people document, test, and write functions

function :: a mathematical object that consumes inputs and produces an output

purpose statement :: a concise, detailed description of what a function does with its inputs

The Design Recipe 10 minutes

Overview

In this lesson students build on what they already know about multiple representations of functions (contracts, examples and definitions) to write purpose statements and gain fluency with the Design Recipe.

Launch

Have students turn to Creating Contracts From Examples (Page 47) and write contracts for the examples provided.

Investigate

You've started to master most of the steps of the Design Recipe, but there's one part you haven't seen yet: *writing a purpose statement* . On its own, a contract is not enough information to generate examples and write a function definition. We need to know what the ***function*** is supposed to do with what it takes in. Programmers and Mathematicians alike find it helpful to restate a problem in their own words. After all, if you can't explain a problem to someone else, you probably don't understand it yourself!

Turn to Writing Examples from Purpose Statements (Page 48) and read the purpose statements. What do you notice? What do you wonder?

Debrief the notice and wonder as a class. Then have students write examples that satisfy each of the contracts and purpose statements on Writing Examples from Purpose Statements (Page 48). OPTIONAL: For more practice, have students complete Writing Examples from Purpose Statements 2.

Synthesize

What are the important elements of purpose statements? Why are purpose statements useful?

Writing Linear Functions 25 minutes

Overview

Students are given a non-working program, which uses a linear function to determine the height of a rocket after a given length of time. The "broken" code is provided to lower cognitive load, allowing students to focus on comprehension (reading the code) and making use of structure (identifying where it's broken).

Launch

Students should have their workbook, pencil, and be logged into WeScheme on their computer. Ask students to open the rocket-height Starter File (Wescheme) and click "Run". By typing `(start rocket-height)`, they will see the simulation start to run on their computer.

Notice and Wonder

What do you notice about this program? What do you wonder?

Survey the class on their "Notices" and "Wonders" and record on the board before moving on to the discussion.

- Is `rocket-height` working?
- Why do you think it's not working?
- What do you think the **purpose** of this function is? How do you know?
- What is the domain of `rocket-height`? *Number*
- What is the range of `rocket-height`? How do you know? *Number, we can tell by looking at the **contract** for the function.*
- As the program is currently written, what happens when I give the function an input of 5? 15? One million? *It always returns 0.*

Investigate

Let's use the Design Recipe to fix `rocket-height`, and get comfortable with writing **purpose statements** .

Have students turn to Word Problem: rocket-height (Page 49), read the problem statement with their partner and write down the ***Contract*** and ***purpose statement***. Then, given the contract and purpose statement, write two examples of how `rocket-height` should work after two different lengths of time.

- Circle and label what's changing in the two examples, just as you did with the green triangle function before.
- Choose a good variable name for what's changing.
- Write the function definition using the variable name.

Once the Design Recipe has been completed in the workbook, students can type the code into the `rocket-height` program, replacing any incorrect code with their own code.

Synthesize

- What was the problem?
- What mistake(s) did the programmer make?
- Where in the Design Recipe did they first go astray?

The Design Recipe allows us to trace mistakes back to the source!

More Interesting Functions flexible

Overview

For teachers who cover quadratic and exponential functions, this activity deepens students' understanding of functions and extends the Design Recipe to include those. This can also be a useful activity for students who finish early, or who need more of a challenge.

Launch

Now that `rocket-height` is working correctly, explore the rest of the file and try the following:

- Remove the comment from before the `(start rocket-height)` and test the program.
- Put the comment back in front of `(start rocket-height)`, remove the comment from `(graph rocket-height)`, and test the program.
- Try out `(space rocket-height)`
- Try out `(everything rocket-height)`

Investigate

- Can you make the rocket fly faster? Slower?
- Can you make the rocket sink down instead of fly up?
- Can you make the rocket *accelerate over time*, so that it moves faster the longer it flies?
- Can you make the rocket blast off *and then land again*?
- Can you make the rocket blast off, *reach a maximum height of exactly 1000 meters*, and then land?
- Can you make the rocket blast off, reach a maximum height of exactly 1000 meters, and then land after exactly 100 seconds?
- Can you make the rocket fly to the edge of the the universe?

Synthesize

Debrief - what did students try? Have students share their experiments with one another!

Additional Exercises:

- Writing Examples from Purpose Statements 2
- Do Examples Have the Same Contracts?
- Do Examples Have the Same Contracts? (2)
- Matching Contracts and Examples
- Matching Contracts and Examples 2

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

Restating the Problem

Restating the Problem

(Also available for Pyret)

Students apply their skills in using the Design Recipe and writing purpose statements to a variety of word problems.

Lesson Goals	Students will be able to: • Understand how to use the ***Design Recipe*** to break down simple word problems. • Create a strong ***purpose statement*** that details in their own words what the ***function*** is doing.

Student-Facing Lesson Goals	• I can use the Design Recipe to break down word problem when writing a function. • I can identify ***domain*** and ***range*** and other quantities in a word problem when writing a function. • I can create and revise a strong purpose statement that explains what the function is doing.
Materials	• Lesson Slides • Purpose Statement organizer (Page 50) • Design Recipe: Direct Variation (Page 51) • Design Recipe Practice (1) (Page 52) • Design Recipe Practice (2) (Page 53) • Design Recipe Practice (3) (Page 54) • Design Recipe: Linear Relationships (1) (Page 55) • Design Recipe: Linear Relationships (2) (Page 56) • Design Recipe: Geometry - Rectangles (Page 57)
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs
Key Points for the Facilitator	• The purpose statement, like the contract, is a comment - something that the computer doesn't read. It's important for readability of their code - there may be other people looking at their code and using their functions! • The domain and range of a function are described as ***data types***, such as Number, String, or Image. • If students struggle with getting started, encourage them to start with one example and use the Mapping examples with Circles of Evaluation organizer. • This activity can work well as a formative review. • This activity is a good time to get students working with someone other than their usual coding partner.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version

Glossary

data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

debug :: to find and fix errors in one's code

design recipe :: a sequence of steps that helps people document, test, and write functions

domain :: the type or set of inputs that a function expects

function :: a mathematical object that consumes inputs and produces an output

purpose statement :: a concise, detailed description of what a function does with its inputs

range :: the type or set of outputs that a function produces

Focusing on Purpose Statements 30 minutes

Overview

This lesson is all about *practice with word problems* , focusing on the specific skill of writing a good purpose statement. Students practice with the Design Recipe and writing quality Purpose Statements. This can be done with their usual coding partner, a new partner, a station review, or another format that suits the class.

Launch

Students should have their workbook, pencil, and be logged into WeScheme on their computer. Students will use the Purpose Statement organizer (Page 50) and the Design Recipe worksheets to work through different practice problems from workbook.

Strategies for Reading Comprehension

MLR 6: 3 Reads - In pairs, the word problem is read 3 times. Students will document their work in the "3 Reads/Stronger & Clearer" handout.

- 1st Read: Teacher reads the word problem. Without any pencil or pen, students discuss: What is the problem about?
- 2nd Read: Partner A reads. Students discuss: What are the quantities?
- 3rd Read: Partner B reads. What is a good purpose statement?

MLR 1: Stronger and Clearer Each Time - Using the "3 Reads + Stronger & Clearer" handout, students will switch partners 3 times.

- Response 1: Write (and/or draw!) your understanding of the word problem.
- Structured Meetings: Meet with another student, and share 1st drafts. Ask clarifying questions and make suggestions of one another, taking notes (repeat with additional meetings as necessary).
- Response 2: Write a second draft, demonstrating your understanding of the word problem.

Students may choose to use the programming environment to test out their functions or experiment with different strategies. Encourage students to try different strategies and ***debug*** their own programs as much as possible.

- **What strategies did you find the most helpful in solving these problems?** *Encourage student discussion while making notes of preferred strategies on the board.*
- **Did any groups disagree on how to solve a problem? What did you do to resolve this?**
- *How can reading a word problem three times help you? *Helps you to slow down and comprehend, makes time to look for information, gives you a chance to catch something you missed the first time, etc.*
- **Where else can you use the strategies we practiced today?**

Investigate

Have students break into teams of 2-4, and use the Design Recipe to solve at least three word problems. We recommend using some of the sample word problems provided in the workbook (see below), but you can also grab any word problem from your math book in which students must define a functional relationship.

- Design Recipe: Direct Variation (Page 51)
- Design Recipe Practice (1) (Page 52)
- Design Recipe Practice (2) (Page 53)
- Design Recipe Practice (3) (Page 54)
- Design Recipe: Linear Relationships (1) (Page 55)
- Design Recipe: Linear Relationships (2) (Page 56)
- Design Recipe: Geometry - Rectangles (Page 57)
- Design Recipe: Geometry - Rectangular Prisms (Page 58)
- Design Recipe: Geometry - Circles (Page 59)
- Design Recipe: Geometry - Cylinders (Page 60)

There are more pages of problems that focus on geometry and linear functions in the additional exercises section at the end of this lesson.

Optional: Ask students to create their own appropriately challenging word problem (with a solution) and collect the responses for later use as "Do Now" tasks or formative assessment.

Synthesize

Which step in the Design Recipe are students feeling the most confident about? The least? At this stage, it is normal for students to feel most confident about the Contract and Examples, and the least confident about Purpose Statements and Definitions.

Design Recipe Games 20 minutes

Overview

The Design Recipe is essentially a systematic way to formalize an unstructured word problem into a structured solution, and each phase formalizes it more than the one that came before it. These activities help students focus on the rigor of each step, and the way those steps are connected. The strategies introduce here can be used in later lessons, and we strongly recommend using at least one of them for every subsequent lesson!

Launch

The Design Recipe makes it possible to solve a problem in pieces, and to *see how those pieces fit together* . For hard problems, knowing how the parts fit together will let you use each step to help you write the next one.

These two activities will involve relatively easy word problems, so the challenge *isn't about solving them!* It's figuring out how the pieces fit together and making sure all of the solutions make sense. Once you know how everything fits together, you'll be able to make fewer mistakes - and even check your work when you do!

Investigate

Design Recipe Telephone

1. Divide the class into groups of three.
2. Choose three word problems (*we'll call them Problems A, B and C*) to give to each group. You can use ones from your textbook, or any of the practice word problems in the workbook that students haven't solved before.
3. In every group, each student is given their own word problem. Student 1 writes the Contract and Purpose for Problem A, Student 2 writes the Contract and Purpose for Problem B, and so on.
4. Once they're done, students should get rid of the word problems by handing them back to the teacher, folding them over, etc. Then they pass their paper to the right so that Student 1 is now looking at the Contract and Purpose for Problem C, Student 2 is looking at the Contract and Purpose for Problem A, and Student 3 is looking at Problem B.

5. Based *solely on the Contract and Purpose*, each student must now write two Examples, as well as circle and label what is changing. If the Contract and Purpose don’t provide enough information, they pass the paper back and the original author has to re-do them.
6. Once they’re done, students get rid of the Contract and Purpose by folding them over. Then they they pass their paper to the right *again*, so that Student 1 is now looking at the Examples for Problem B, Student 2 is looking at the Contract and Purpose for Problem C, and Student 3 is looking at Problem A.
7. Based *solely on the Examples* (and the circles-and-labeled variables), students must derive the function definition. If the Examples don’t provide enough information, they pass the paper back and the original author has to re-do them.

This activity can be repeated several times, or done as a timed competition between teams. The goal is to emphasize that each step - if done correctly - makes the following step incredibly simple.

Where’d You Get That?

Divide the class into pairs, giving each pair two word problems (the whole class can use the same set, or different ones), and have students solve one problem each *independently* . Once finished, students take turns *challenging each other* . The Challenger always starts at the **bottom** of the page, physically pointing to one part of the function definition and asking "where’d you get that?" The Defender has to *physically point* to some location in the Examples, and explain exactly how they got that part of the definition. This is repeated for every other step in the recipe, as students work their way back to the original word problem. For example:

- **Challenger** (pointing at variable in the Definition): Where’d you get that?
- **Defender** (pointing at label in the Examples): Well, I circled the parts of the Examples that change, and gave them that label.
- **Challenger** (pointing at the label): OK, but where did you get the label?
- **Defender** (pointing at Purpose Statement): I used that term in the Purpose Statement.
- **Challenger** (pointing at Purpose Statement): Where’d you get that term?
- **Defender** (pointing to Word Problem): I got it from reading the Word Problem.

Common Misconceptions

Mathematically confident students will *actively resist* these activities, because they may be used to having the answer come to them almost as soon as they finish reading the word problem (this is the same objection those students have to explaining "how they got the answer").

Synthesize

The Design Recipe is a way of slowing down and thinking through each step of a problem. If we already know how to get the answer, why would it ever be important to know how to do each step the slow way?

Sample Responses:

- Someday we won’t be able to get the answer, and knowing the steps will help
- So we can help someone else who is stuck
- So we can work with someone else and share our thinking
- So we can check our work

Additional Exercises

- Design Recipe Practice: Cylinders (Page 60)
- Design Recipe Practice: Surface Area of Prisms (Page 58)
- Design Recipe Practice: Negative Slope (Page 56)
- Design Recipe: Linear Relationships (3)
- Design Recipe: Linear Relationships (4)

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

Functions for Character Animation

Functions for Character Animation

(Also available for Pyret)

Students define functions that control the movement of the target and danger in their games

Lesson Goals	Students will be able to: • Apply the ***Design Recipe*** to create a ***function*** given the constraints of a word problem. • Explain the basics of animation.
Student-Facing Lesson Goals	• I can use the Design Recipe to make a function. • I can describe how animation works.

<table>
<tr><td><b>Materials</b></td><td>• Lesson Slides
• Danger and Target Movement (Page 61)</td></tr>
<tr><td><b>Preparation</b></td><td>• Make sure all materials have been gathered
• Decide how students will be grouped in pairs</td></tr>
<tr><td><b>Key Points for the Facilitator</b></td><td>• Encourage students to take their time in understanding <i>why</i> we want to fix <code>update-danger</code> and <code>update-target</code>.
• Students might be confused as to <i>how</i> the animation is working. The <code>make-game</code> function at the bottom of the file has many inputs - including <code>update-danger</code> and <code>update-target</code>. <code>make-game</code> takes in all those inputs, including the functions we'll write, and creates the interactive window that we see when we press the Run button!</td></tr>
<tr><td><b>Language Table</b></td><td>
<table>
<tr><th>Types</th><th>Functions</th><th>Values</th></tr>
<tr><td>Number</td><td><code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>expt</code>, <code>sqr</code>, <code>sqrt</code></td><td><code>4</code>, <code>-1.2</code>, <code>2/3</code>, <code>pi</code></td></tr>
<tr><td>String</td><td><code>string-length</code>, <code>string-repeat</code>, <code>string-contains?</code></td><td><code>"hello"</code>, <code>"91"</code></td></tr>
<tr><td>Boolean</td><td><code>&lt;</code>, <code>&lt;&gt;</code>, <code>&lt;=</code>, <code>&gt;=</code>, <code>string&lt;?</code>, <code>string&gt;?</code>, <code>string=?</code>, <code>string&lt;&gt;?</code>, <code>string&gt;=?</code></td><td><code>true</code>, <code>false</code></td></tr>
<tr><td>Image</td><td><code>star</code>, <code>triangle</code>, <code>circle</code>, <code>square</code>, <code>rectangle</code>, <code>rhombus</code>, <code>ellipse</code>, <code>regular-polygon</code>, <code>radial-star</code>, <code>text</code>, <code>overlay</code>, <code>above</code>, <code>beside</code>, <code>rotate</code>, <code>scale</code>, <code>flip-horizontal</code>, <code>flip-vertical</code></td><td>🔵🔺🔶</td></tr>
</table>
</td></tr>
</table>

Click here to see the prior unit-based version

Glossary

coordinate :: a number or set of numbers describing an object's location

design recipe :: a sequence of steps that helps people document, test, and write functions

function :: a mathematical object that consumes inputs and produces an output

Animation 45 minutes

Overview

Students connect the behavior of functions with changing coordinate values, ultimately leading to animation.

Launch

Students should have their computer, contracts page, and pencil. Students should have their own **game file** open in a separate window or tab.

- How does a flip-book animation work? *Each page of the book is slightly different, and the pages go so fast that the motion looks smooth.*
- Why do we see movement from still images? *Our eyes fill in the gaps between rapidly changing images.*
- How might this apply to our game? *If we change image **coordinates** a little bit at a time, they will appear to move.*

Draw a number line on the board, running from 0 to 1000 (you can also lay tape on the floor, or use a tile floor as a coordinate plane!). Select 2 student volunteers - one to be `TARGET`, one to be `DANGER`. Start with just `TARGET`.

- Have the class select a starting x- and y-coordinate for the `TARGET`, and have the volunteer move to that position on the number line or coordinate plane.
- The TARGET character moves by 50 (pixels) on each frame of the game.
- When they hear "update target" followed by their current location, the `TARGET` takes a step in the negative direction, moving down the x-axis by 50 (pixels).
- We make TARGET move by calling out `(update-target 300)`, `(update-target 250)`, etc.

How quickly could I get TARGET to move across the classroom?

After practicing with TARGET, add DANGER in.

- `DANGER` takes a step in the *positive direction* when they hear "update danger" followed by their current x-coordinate.
- We make `DANGER` move by calling out `(update-danger 40)`, `(update-danger 39)`, etc.

- On a standard number line, if the `DANGER` is moving to the right, is its x-coordinate increasing or decreasing?

Practice this a few times with your volunteer, asking the class what their new x-coordinate is each time. Then have the other students call the update-danger function.

- **What did you notice about the movement of TARGET and DANGER? What was changing about them?**

  Answers will vary: they were moving horizontally, their x-coordinates were changing, they were not moving smoothly, etc.

- **What jobs could we hand over to the computer to make it possible for us to play the game?** *The computer could handle automatically moving TARGET and DANGER, then we could control the movement of PLAYER.*

Investigate

- Have students examine the `update-danger` function in their Game Starter File, identify the contract, and interpret what the function is currently doing.
- Guide students as they complete the first word problem on Danger and Target Movement (Page 61), and transfer the code to their Game Starter File.

When students press the Run button, the working `update-danger` function should automatically move the `DANGER` image across the screen!

Have students complete the second word problem on Danger and Target Movement (Page 61), with their partner and transfer the code to their Game Starter File. Press Run to see `DANGER` and `TARGET` move across the screen independently!

Extension Activities

Once students have successfully gotten `update-target` and `update-danger` working, they can change the functions to make the characters move in whichever direction and whatever speed they want! They should be sure to modify their purpose statements and examples if they change their functions.
Want 2-D movement? A supplemental lesson linked here provides information on how to modify these functions to allow movement in the x *and* y directions!

Surface Area of a Rectangular Prism

Surface Area of a Rectangular Prism

(Also available for Pyret)

Students define the shapes used to build a rectangular prism, print them, cut them out, and build the rectangular prism. Then they use their model to calculate the surface area.

Lesson Goals	Students will be able to: • Demonstrate understanding of ***surface area*** and use that understanding to calculate surface area of ***rectangular prisms***
Student-facing	

<table>
<tr><td><b>Goals</b></td><td><ul><li>I can explain what <i><b>surface area</b></i> is.</li><li>I can use my understanding of surface area to calculate the surface area of any <i><b>rectangular prism</b></i>.</li></ul></td></tr>
<tr><td><b>Materials</b></td><td><ul><li>Lesson Slides</li><li>This lesson requires a printer, scissors and tape.</li><li>Colored pencils will be useful for some students.</li><li>Prism Starter File (Wescheme)</li></ul></td></tr>
<tr><td><b>Preparation</b></td><td><ul><li>Students will be generating files that need to be printed. If their computers are not connected to a printer, make a plan for how printing will happen.</li><li>Make sure all materials have been gathered.</li><li>Decide how students will be grouped in pairs.</li></ul></td></tr>
<tr><td><b>Key Points For The Facilitator</b></td><td><ul><li>This lesson focuses on developing students' understanding of <i><b>Surface Area</b></i> of prisms, such that they can build their own formulas. Instructors are encouraged not to reference any pre-defined formulas during the exploration.</li></ul></td></tr>
<tr><td><b>Language Table</b></td><td>
<table>
<tr><th>Types</th><th>Functions</th><th>Values</th></tr>
<tr><td>Number</td><td><code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>expt</code> , <code>sqr</code> , <code>sqrt</code></td><td><code>4</code> , <code>-1.2</code> , <code>2/3</code> , <code>pi</code></td></tr>
<tr><td>String</td><td><code>string-length</code> , <code>string-repeat</code> , <code>string-contains?</code></td><td><code>"hello"</code> , <code>"91"</code></td></tr>
<tr><td>Boolean</td><td><code><</code> , <code><></code> , <code><=</code> , <code>>=</code> , <code>string<?</code> , <code>string>?</code> , <code>string=?</code> , <code>string<>?</code> , <code>string>=?</code></td><td><code>true</code> , <code>false</code></td></tr>
<tr><td>Image</td><td><code>star</code> , <code>triangle</code> , <code>circle</code> , <code>square</code> , <code>rectangle</code> , <code>rhombus</code> , <code>ellipse</code> , <code>regular-polygon</code> , <code>radial-star</code> , <code>text</code> , <code>overlay</code> , <code>above</code> , <code>beside</code> , <code>rotate</code> , <code>scale</code> , <code>flip-horizontal</code> , <code>flip-vertical</code></td><td>🔵🔺🔶</td></tr>
</table>
</td></tr>
</table>

Glossary

face :: the shapes on the outside of a figure

rectangular prism :: a solid figure which has 6 faces, all of which are rectangular

surface area :: the sum of the areas of the faces of a polyhedron or the total area that the surface of a solid object occupies

Surface Area

Overview

Students build on their experience with writing code to define shapes. They will define shapes for all of the ***faces*** of a ***rectangular prism***, print them, cut them out, and build the rectangular prism. Then they will use their model to calculate the surface area and write code to do the same.

Launch

Students should open the Prism Starter File (Wescheme) in their browser and click run.

Click run and type `prism` into the Interactions Area to see an image of a rectangular prism. Notice that the ***faces*** and dimensions (Length, Width, and Height) are labeled.

> Faces are the shapes on the outside of the figure. Edges are the line segments where the faces of solid figures meet in each of the three dimensions.

- How would you describe the faces of this prism?
- Skip down to the code on line 19. It reads `(define lst (list front))`, which defines `lst` to be a *list* of values. This list will include all of the faces of the prism, bu right now it only includes `front`. Add the names of each of the remaining faces to the list. (Order doesn't matter.)
- Once you complete your list, go back up to line 17 and look at the definition for `front`. Type `front` into the interactions area. What do you get?
- Just as `front` has been defined to draw a rectangle whose dimensions are `width` and `height`, you will need to write definitions for each of the other faces of the prism the you put on your list. Start adding definitions on line 18 and add a line for each definition so that all of the faces are defined between `front` and `lst`. Click run and test each of them in the interactions area to make sure that they match the prism you started with.

Facilitation Note

The sample definition was written to make the image of an outlined rectangle with a black and white printer in mind. If you have access to a color printer in your classroom, you may want students to change the code of `front` to better match what they see in the image of `prism` and code the remaining faces with solid rectangles to match the image they are looking at. If you do not have access to a color printer, but think that colors would support your students, you can have them color the rectangles on the printout before cutting and assembling the prism.

- When you've finished, click Run and type `(print-imgs lst)`. What do you Notice? What do you Wonder?
- Do you have enough shapes to cover all of the faces of the prism?
- Read the comments in the file to learn how to customize and print the faces to build your prism.

Investigate

Have students cut out and tape together the images they defined to form a 3-dimensional paper model of a rectangular prism.
Students will then use their models to calculate the surface area.

Supporting students with learning variations

- Labeling the shapes with face names and/or area before taping them together may help some students.
- Printing two copies of the file (one to cut and one to write on) might best support other students.

- Once you've built your prism, use it to help you calculate the surface area of the figure. Then, turn to task 2 in the Prism Starter File and define surface area to calculate the area of the prism using width, length and height.

Synthesize

Have students share the codes they wrote to define surface area. Did students all write the code the same way? Can anyone see other ways that they could have written the code?

- How did building the prism help you to understand surface area?
- How did writing the code for surface area help you to understand surface area?

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

Problem Decomposition

Problem Decomposition

(Also available for Pyret)

Students take a closer look at how functions can work together by investigating the relationship between revenue, cost, and profit.

Lesson Goals	Students will be able to: • Write a ***function*** that explicitly uses another function. • Explain the benefits and drawbacks of functions that depend on each other. • Explain the difference between bottom-up and top-down strategies.
Student-Facing Lesson Goals	• I can explain the benefits and drawbacks of functions that use other functions. • I can write a function that uses another function.
Materials	• Lesson Slides

- revenue (Page 63)
- profit (Page 64)

Preparation

- Make sure all materials have been gathered
- Decide how students will be grouped in pairs

Key Points for the Facilitator

- There are several ways to write the `profit` function - use this opportunity for discussion and to promote higher-order critical thinking.
- If students are struggling with understanding the basics of the problem, start by coming up with examples of `cost` and `revenue`. If Sally sells one glass, what is her total revenue? How much does it cost her to produce that one glass?
- Ensure students understand the difference between "revenue" and "profit", and that "cost" refers to the cost of *making* the lemonade, not the amount Sally is charging.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Glossary

function :: a mathematical object that consumes inputs and produces an output

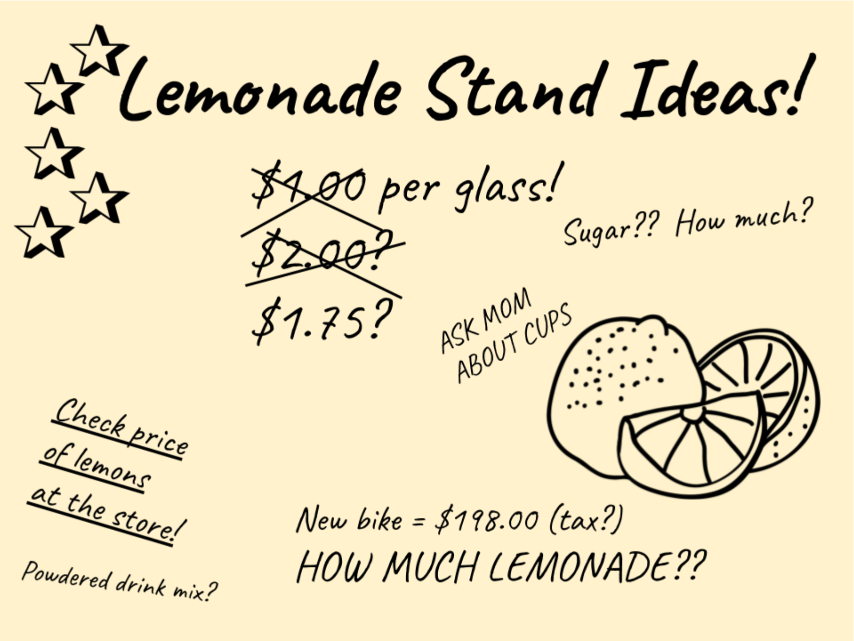
Problem Decomposition 30 minutes

Overview

Students are introduced to word problems that can be broken down into *multiple* problems, the solutions to which can be composed to solve other problems. They adapt the Design Recipe to handle this situation.

Launch

Students should have their workbook, pencil, and be logged into WeScheme and have their workbooks with a pen or pencil.
Display the following image:



Notice and Wonder

Have students share everything they notice about the situations described above. Then, separately, have them share what they wonder.

One example of a *relationship* we can find in this situation is that Sally takes in $1.75 for every glass she sells: $revenue = \$1.75 \times glasses$

What other relationships can you find here?

(Give students a chance to discuss and brainstorm)

- Every glass sold brings in $1.75 in **revenue**
- Every glass sold costs $0.30 in **costs**, such as lemonds, sugar and water
- Every glass sold brings in some amount of **profit**: it costs a certain amount to make, but it brings in another amount in revenue

Investigate

Students form groups and brainstorm their ideas for functions. Students can use any strategies they've learned so far.

Strategies for English Language Learners

MLR 7 - Compare and Connect There are several correct ways to write the functions needed for Sally's Lemonade. Have students compare methods and develop understanding and language related to mathematical representation and methods. What are the advantages of the different solutions? What are some drawbacks?

- **What is the difference between *revenue* and *profit*?** *Revenue is the total amount of money that comes in, profit is the remaining money after cost has been subtracted.*
- **How could Sally *increase* her profits?** *By decreasing her costs, raising her prices (which increases revenue), by selling more lemonade.*
- **What is the *relationship* between profit, cost, and revenue?** *Profit = Revenue - Cost*

Students work with their partners to develop their function models for revenue (Page 63), and profit (Page 64), using the Design Recipe.

While students are working, walk the room and gauge student understanding. There is more than one correct way to write the `profit` function! Encourage discussion between students and push students to develop their thinking on the advantages and disadvantages of each correct solution.

Synthesis

This activity started with a situation, and students modeled that situation with functions. One part of the model was *profit*, which can be written several ways, for example:

```
(define (profit g) (- (* 1.75 g) (* 0.30 g)))
(define (profit g) (* (- 1.75 0.30) g))
(define (profit g) (* 1.45 g))
(define (profit g) (- (revenue g) (cost g)))
```

- Which way is "best", and why?
- If lemons gets more expensive, which way requires the least amount of change?
- If sugar gets less expensive, which way requires the least amount of change?

Big Ideas

1. `profit` can be *decomposed* into a simple function that uses the `cost` and `revenue` functions.
2. Decomposing a problem allows us to solve it in smaller pieces, which are also easier to test!
3. These pieces can also be re-used, resulting in writing less code, and less *duplicate* code.
4. Duplicate code means more places to make mistakes, especially when that code needs to be changed.

Top-Down vs. Bottom-Up 20 minutes

Overview

Students explore problem decomposition as an explicit strategy, and learn about two ways of decomposing.

Launch

Top-Down and *Bottom-Up* design are two different strategies for problem decomposition.

Bottom-Up: start with the small, easy relationships first and then build our way to the larger relationships. In the Lemonade Stand, you defined `cost` and `revenue` first, and then put them together in `profit`.

Top-Down: start with the "big picture" and then worry about the details later. We could have started with `profit`, and made a to-do list of the smaller pieces we'd build later

Investigate

Consider the following situation:

Jamal's trip requires him to drive 20mi to the airport, fly 9,000mi, and then take a bus 6mi to his hotel. His average speed driving to the airport is 40mph, the average speed of an airplane is 575mph, and the average speed of his bus is 15mph.

Aside from time waiting for the plane or bus, how long is Jamal in transit?

This can be decomposed via Top-Down or Bottom-Up design. What functions would you define to solve this, and in what order? For extra credit, you can actually compute the answer!

Synthesize

Make sure that students see *both* strategies, and have them discuss which they prefer and why.

Simple Inequalities

Simple Inequalities

(Also available for Pyret)

Students discover the `Boolean` data type. They then learn to identify solutions and non-solutions of inequalities.

Lesson Goals	Students will be able to: • Describe the solution set of a simple inequality
Student-Facing Lesson Goals	• I can find solutions and non-solutions to an inequality
Materials	• Lesson Slides (Google) • Boolean Functions (Page 66) • Simple Inequalities (Page 67) • Lesson Slides (Google) • Boolean Starter File (Wescheme) • Simple Inequalities Starter File. (Wescheme) • Optional: *Word Problem: hot?*
Preparation	• Make sure all materials have been gathered

<table>
<tr><td></td><td><ul><li>Decide how students will be grouped in pairs</li></ul></td></tr>
<tr><td><b>Supplemental Resources</b></td><td><ul><li>Booleans Review</li></ul></td></tr>
<tr><td><b>Key Points for the Facilitator</b></td><td><ul><li>A <b><i>Boolean</i></b> is just another <b><i>data type</i></b>, like Numbers or Images. But unlike those types, there are only <i>two</i> values: <code>true</code> and <code>false</code>. While simple to explain, this different behavior can be confusing for some students.</li><li>Boolean-producing functions are essentially yes-or-no questions, so the names of the functions in this lesson read like questions.</li><li>For example, <code>odd?</code> and <code>even?</code> (pronounced "odd-huh?" and "even-huh?" in Scheme) are both functions that ask if a given number is odd or even.</li><li>Role-playing can help students understand the jobs of <code>odd?</code> and <code>even?</code>.</li></ul></td></tr>
<tr><td><b>Language Table</b></td><td>
<table>
<tr><th>Types</th><th>Functions</th><th>Values</th></tr>
<tr><td>Number</td><td><code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>expt</code>, <code>sqr</code>, <code>sqrt</code></td><td><code>4</code>, <code>-1.2</code>, <code>2/3</code>, <code>pi</code></td></tr>
<tr><td>String</td><td><code>string-length</code>, <code>string-repeat</code>, <code>string-contains?</code></td><td><code>"hello"</code>, <code>"91"</code></td></tr>
<tr><td>Boolean</td><td><code><</code>, <code><></code>, <code><=</code>, <code>>=</code>, <code>string<?</code>, <code>string>?</code>, <code>string=?</code>, <code>string<>?</code>, <code>string>=?</code></td><td><code>true</code>, <code>false</code></td></tr>
<tr><td>Image</td><td><code>star</code>, <code>triangle</code>, <code>circle</code>, <code>square</code>, <code>rectangle</code>, <code>rhombus</code>, <code>ellipse</code>, <code>regular-polygon</code>, <code>radial-star</code>, <code>text</code>, <code>overlay</code>, <code>above</code>, <code>beside</code>, <code>rotate</code>, <code>scale</code>, <code>flip-horizontal</code>, <code>flip-vertical</code></td><td>🔵🔺🔶</td></tr>
</table>
</td></tr>
</table>

Glossary

Boolean :: a type of data with two values: true and false

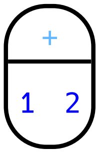
data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

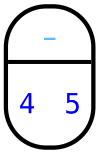
Introducing Booleans 20 minutes

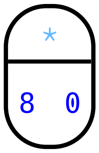
Launch

Students should be logged into WeScheme

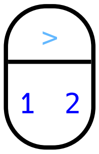
Ask students to evaluate Circles of Evaluation for simple expressions they've seen before, and ask them to convert them into code.

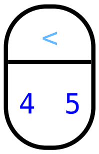
- 

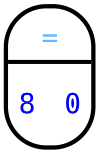
- 

- 

Then show them unfamiliar Circles of Evaluation, and ask them to hypothesize what they think they mean, what they will evaluate to, and what the code would look like.

- 

- 

- 

Have students convert these Circles to code and type them in. What did they evaluate to? What do they think the outputs mean?

Values like `true` and `false` obviously aren't Numbers or Images. But they also aren't Strings, or else they would have quotes around them. We've found a *new data type*, called a ***Boolean***.

Investigate

Have students open the Boolean Starter File (Wescheme)

- Explore the five functions in this starter file: `odd?`, `even?`, `less-than-one?`, `continent?`, `primary-color?`
- All five functions produce ***Booleans***. Through your exploration, see if you can come up with an explanation of what a ***Boolean*** is.

A ***Boolean*** is just another ***data type***, like Numbers or Images. But unlike the others there are only two values: `true` and `false`.

- Turn to Boolean Functions (Page 66) and use the starter file to complete the questions, identifying inputs that will make each function produce `true`, and inputs that will make each function `false`.

Synthesize

- Students will see functions on this page that they've never encountered before! But instead of answering their questions, encourage them to make a *guess* about what they do, and then type it in to discover for themselves.
- Explicitly point out that *everything they know still works!* They can use their reasoning about Circles of Evaluation and Contracts to figure things out.

Common Misconceptions

- Many students - especially traditionally high-achieving ones - will be very concerned about writing examples that are "wrong." The misconception here is that an expression that produces `false` is somehow *incorrect*. You can preempt this in advance, by explaining that our Boolean-producing functions *should sometimes return false*.

Introducing Inequalities 20 minutes

Overview

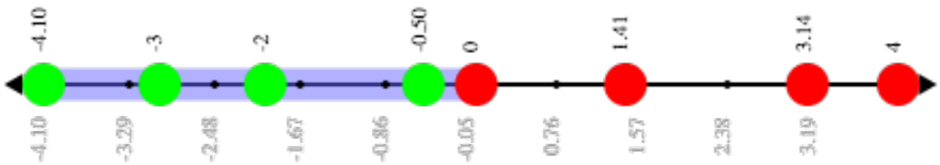
Students discover (or expand their understanding of) inequalities by identifying solutions and non-solutions and connecting expressions to graphic representations.

Launch

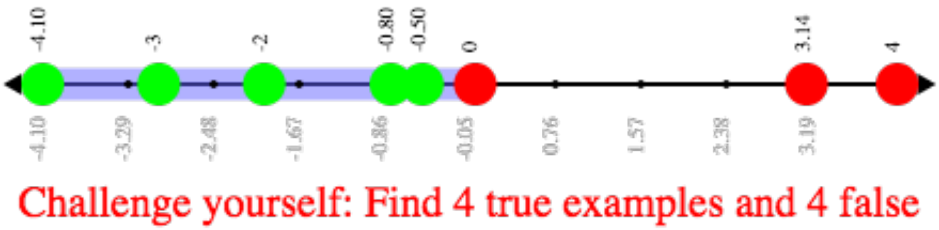
Have students open the [Simple Inequalities Starter File. (Wescheme)]()
Equations typically have finite solution sets: there’s only one answer for an unknown, or perhaps several answers. Inequalities, on the other hand, can have *infinite* solutions. Inequality expressions divide all of the numbers in the universe into two categories: solutions and non-solutions. It is important that students are able to recognize that there are many possible solutions and non-solutions to an inequality and that they can identify whether or not a given number is or isn’t part of the solution set. This starter file includes a special `inequality` function that takes in a function, *which tests numbers in an inequality* , a list of 8 numbers *(to test in the function)* , and plots the numbers and a graph of the inequality on a number line.

> The solution set is shaded in blue, with points shaded green (solution) and red (non-solution).

The resulting plot shows the number line, with all solution values shaded in blue. The 8 numbers provided in the list are shown as green (solution) or red (non-solution) circles. A successful input will include 4 solutions and 4 non-solutions, so the image returned will show 4 green dots and 4 red dots.



If their list of 8 values doesn’t include an equal number of solutions and non-solutions there will be an unequal distribution of red and green dots and they will get an error message encouraging them to adjust their list.



Encourage students to use negatives, positives, fractions and decimals as they generate their lists.

The starter file includes an example. Read the example code in the file carefully and click run to see the image it returns. Discuss the code with your partner.

- What do you Notice?
- What do you Wonder?

Hiding Example Code

In order to stop seeing the examples written into the starter file code, students can comment out the example code by adding a # in front of each of the lines they want to hide.

Investigate

Have students open to the Simple Inequalities (Page 67) and complete it with a partner, identifying solutions and non-solutions to each inequality and testing them in the Simple Inequalities Starter File. (Wescheme)

Synthesize

- What patterns did you observe in how the inequalities worked?

Additional Exercises:

- Word Problem: hot?

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Compound Inequalities: Solutions & Non-Solutions

Compound Inequalities: Solutions & Non-Solutions

(Also available for Pyret)

Students build upon their understanding of booleans and simple inequalities to compose compound inequalities using the concepts of union and intersection.

Lesson Goals	Students will be able to: • Understand how the conjunctions `and` and `or` differ • Describe how functions can work together • Describe the solution set of a compound inequality
Student-Facing Lesson Goals	• I can explain the difference between how `and` and `or` are used in programming • I can use two or more inequalities together and describe the area they enclose

Materials

- Lesson Slides
- Converting Circles of Evaluation to Code (Page 68)
- Compound Inequalities — Practice (Page 69)
- Compound Inequalities: Solutions & Non-Solutions (Page 70)
- Compound Inequality Functions (Page 71)
- Compound Inequalities Starter File (Wescheme)
- *Optional: Converting Circles of Evaluation with Booleans to Code 2*

Preparation

- Make sure all materials have been gathered
- Decide how students will be grouped in pairs

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `>`, `<>`, `<=`, `>=`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`, `and`, `or`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Glossary

function :: a mathematical object that consumes inputs and produces an output

intersection :: the set of values that makes both inequalities true

union :: the set of values that makes either or both of a set of inequalities true

Introducing Compound Inequalities

Overview

Students consider the need to *compose* inequalities, and think about how to write them.

Launch

We use inequalities for lots of things:

- Is it hot out? ($temperature > 80°$)
- Did I get paid enough for painting that fence? ($paid \geq \$100$)
- Are the cookies finished baking? ($timer = 0$)

Have students come up with other examples.

Many times we need to *combine* inequalities:

- Should I go to the beach? ($temperature > 80°$ and $weather = "\ sunny\ "$)
- Was this burrito worth the price? ($taste = "\ delicious\ "$ and $price \leq \$15$)

Have students come up with other examples.

Guide students through other examples of `and` and `or` with various statements, such as "I'm wearing a red shirt AND I'm a math teacher, true or false?" or "I'm an NBA basketball star OR I'm having pizza for lunch, true or false?". This can make for a good sit-down, stand-up activity, where students take turns saying compound boolean statements and everyone stands if that statement is true for them.

Investigate

Both mathematics and programming have ways of combining - or *composing* - inequalities.

Complete Converting Circles of Evaluation to Code (Page 68) and Compound Inequalities — Practice (Page 69).

Synthesize

Be really careful to check for understanding here.

- Expressions using `and` only produce `true` if both of their sub-expressions are `true`.
- Expressions using `or` produce `true` if **either** of their sub-expressions are `true`.

Strategies for English Language Learners

When describing compound inequalities, be careful not to use "english shortcuts". For example, we might say "I am holding a marker *and* an eraser" instead of "I am holding a marker *and* I am holding an eraser." These sentences mean the same thing, but the first one obscures the fact that "and" joins two complete phrases. For ELL/ESL students, this is unecessarily adds to cognitive load!

Solutions and Non-Solutions of Compound Inequalities

Launch

Have students identify 4 solutions and 4 non-solutions for each of the following inequalities.

- $x > 5$
- $x \leq 15$

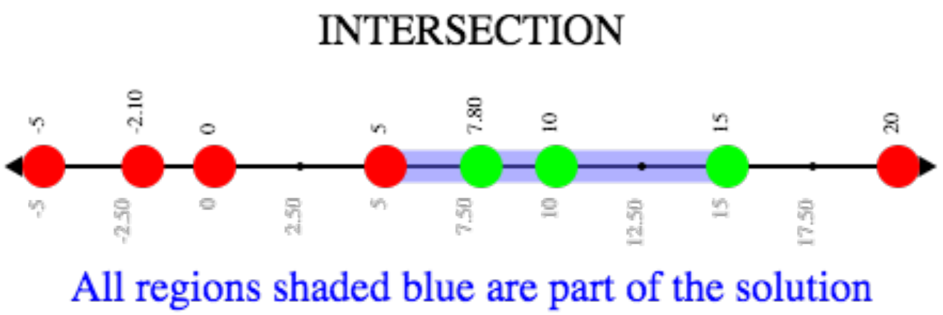
What about the solution set of $x > 5$ `and` $x \leq 15$? What numbers make both of these inequality expressions true?
How would that be different from $x > 5$ `or` $x \leq 15$? What numbers make at least one of these inequality expressions true?

Investigate

Have students log in to the

Compound Inequalities Starter File (Wescheme), read the code carefully and click run to see the graphs they've just considered.
This starter file includes two special functions.
`and-intersection` takes in two functions and a list of numbers and produces a graph with the points and the shaded ***intersection*** of values that make both of the inequalities true.

```
(define (gt5 x) (> x 5))
(define (lte15 x) (<= x 15))
(and-intersection gt5 lte15 (list -5 -2.1 0 5 10 39/5 15 20))
```

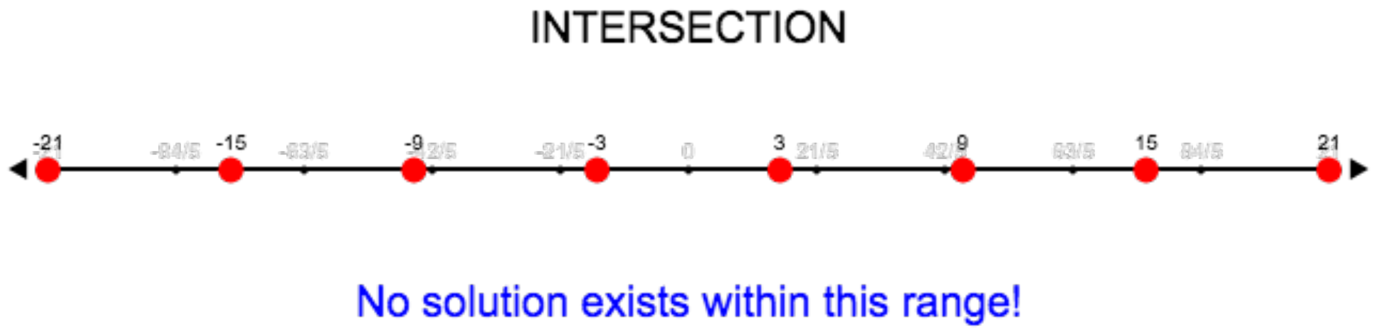

INTERSECTION

All regions shaded blue are part of the solution

```
(define (lt5 x) (< x 5))
```

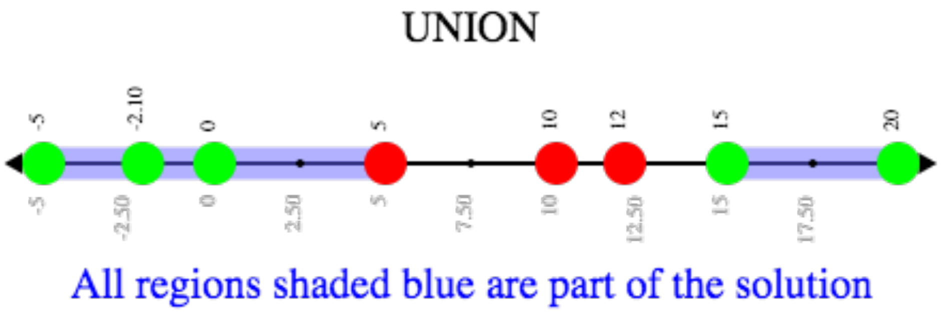
```
(define (gte15 x) (>= x 15))
(and-intersection lt5 gte15 (list -5 -2.1 0 5 10 39/5 15 20))
```

Note: Some pairs of inequalities do not intersect at all and therefore have ***no solutions*** *.*



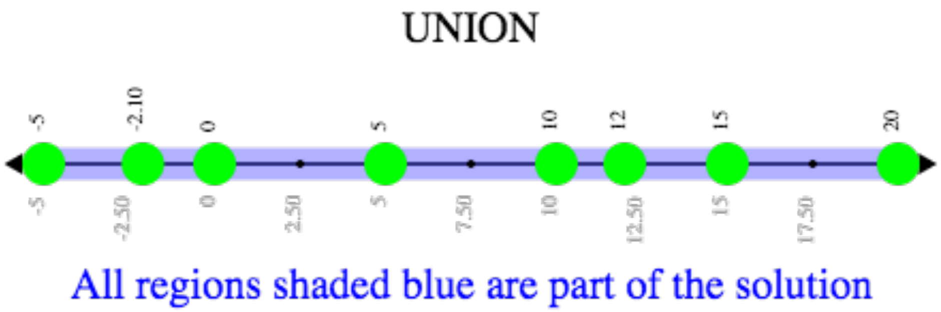
`or-union` takes in two functions and a list of numbers and produces a graph with the points and the shaded ***union*** of values that make either or both of the inequalities true.

```
(define (lt5 x) (< x 5))
(define (gte15 x) (>= x 15))
(or-union lt5 gte15 (list -5 -2.1 0 5 10 39/5 15 20))
```



```
(define (gt5 x) (> x 5))
(define (lte15 x) (<= x 15))
(or-union gt5 lte15 (list -5 -2.1 0 5 10 39/5 15 20))
```

Note: Some ***unions****, like the one below, include* ***all real numbers*** *; they have have* ***infinite solutions*** *that satisfy at least one of the inequalities.*



Turn to Compound Inequalities: Solutions & Non-Solutions (Page 70) and explore the compound inequalities listed using the Compound Inequalities Starter File, identifying solutions and non-solutions for each.

Instead of defining two functions as simple inequalities, we could produce the same graph by defining one function to be a compound inequality.

```
(define (fiveto15 x) (and (> x 5) (<= x 15)))
(inequality fiveto15 (list -5 -2.1 0 5 10 12 15 20))
```

Turn to Compound Inequality Functions (Page 71) and have students write code to describe the compound inequalities pictured.

Synthesize

- How did the graphs of intersections and unions differ?

Additional Exercises:

- Converting Circles of Evaluation with Booleans to Code 2

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Sam the Butterfly - Applying Inequalities

Sam the Butterfly - Applying Inequalities

(Also available for Pyret)

Students will apply their understanding of inequalities to keep game characters on screen.

Lesson Goals	Students will be able to: apply their understanding of inequalities to keep a game character on the screen
Student-Facing Lesson Goals	I can use what I know about inequalities to define the boundaries that will keep a character on the screen.
Materials	Lesson Slides Introducing Sam (Page 72) Left and Right (Page 73) Word Problem: onscreen? (Page 74)

	Sam The Butterfly starter file (Wescheme) Optional: *Keeping NinjaCat in the Game*
Preparation	Make sure all materials have been gathered Decide how students will be grouped in pairs
Supplemental Resources	
Language Table	

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `>`, `<>`, `<=`, `>=`, `string-equal`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`, `and`, `or`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Glossary

coordinate :: a number or set of numbers describing an object’s location

expression :: a computation written in the rules of some language (such as arithmetic, code, or a Circle of Evaluation)

Introducing Sam 30 minutes

Overview

Students are introduced to Sam the Butterfly: a simple activity in which they must write 1-step inequalities to detect when Sam has gone too far in one dimension.

Launch

Have students open the
Sam The Butterfly starter file (Wescheme) and click "Save."
Have students turn to the Introducing Sam (Page 72), click run and use the arrow keys to investigate the program with their partner.

- **What is something you noticed about this program?** *As Sam moves, the **coordinates** are displayed at the top of the screen; the coordinates are all in the 1st quadrant; etc.*
- **What do you see when Sam is at (0,0)? Why is that?** *You only see part of Sam's wing. Sam's position is based on the center of Sam's image.*
- **How far can Sam go to the left and stay on the screen?** *Up to, but not beyond, an x of -40.*
- **How could we write this as an *expression*?** $x \geq -40$, or $x > -50$

Every time Sam moves, we want to check and see if Sam is safe.

- There are three functions defined in this file. What are they?

Note: In this programming language, question marks are prounced "huh?". So `safe-left?` would be prounounced "safe left huh?" This can be a source of some amusement for students!

Optional: For extra scaffolding...

- **What *should* our left-checking function do?** *Check to see if x is greater than -50*
- **What *should* our right-checking function do?** *Check to see if x is less than 690*
- **What should `onscreen?` do?** *Answers may vary, let students drive the discussion, and don't give away the answer*

Investigate

With their partners, students complete Left and Right (Page 73). Once finished, students can fix the corresponding functions in their Sam the Butterly file, and test them out.
Students will notice that fixing `safe-left?` keeps Sam from disappearing off the left, but fixing `safe-right?` doesn't seem to keep Sam from disappearing off the right side! When students encounter this, encourage them to look through the code to try and figure out why. The answer will be revealed in the next lesson.

- Recruit three new student volunteers to roleplay those same functions, which have now been *corrected*. Make sure students provide correct answers, testing both `true` and `false` conditions using coordinates where Sam is onscreen and offscreen.

Common Misconceptions

- Many students - especially traditionally high-achieving ones - will be very concerned about writing examples that are "wrong." The misconception here is that an expression that produces `false` is somehow *incorrect*. You can preempt this in advance, by explaining that our Boolean-producing functions *should sometimes return false*, such as when Sam is offscreen.
- Push students to think carefully about corner-cases, such as when Sam is *exactly* at -50 or 690.

Protecting Sam on Both Sides 30 minutes

Overview

Students solve a word problem involving compound inequalities, using `and` to compose the simpler Boundary-checking functions from the previous lesson.

Launch

Note: In this programming language, question marks are prounced "huh?". So `safe-left?` would be prounounced "safe left huh?" This can be a source of some amusement for students!

- Recruit three student volunteers to roleplay the functions `safe-left?`, `safe-right?` and `onscreen?` . Give them 1 minute to read the contract and code, as written in the program.
- As in the previous lesson, ask the volunteers what their name, Domain and Range are, and then test them out by calling out their name, followed by a number. (For example, "(safe-left? 20)!", "(safe-right? -100)!", "(onscreen? 829)!") **Note"** the code for `onscreen` *calls the safe-left function!*. So the student roleplaying `onscreen` should turn to `safe-left` and give the input to them.

For example:

- **Facilitator**: "onscreen-huh 70"
- **onscreen?** (turns to safe-left?): "safe-left-huh 70"
- **safe-left?**: "true"
- **onscreen?** (turns back to facilitator): "true"

- **Facilitator**: "onscreen-huh -100"
- **onscreen?** (turns to safe-left?): "safe-left-huh -100"
- **safe-left?**: "false"
- **onscreen?** (turns back to facilitator): "false"

- **Facilitator**: "onscreen-huh 900"
- **onscreen?** (turns to safe-left?): "safe-left-huh 900"
- **safe-left?**: "true"

- **onscreen?** (turns back to facilitator): "true"

Ask the rest of the class

- What is the problem with `onscreen?`?
  It’s only talking to `safe-left?`*, it’s not checking with* `safe-right?`
- How can `onscreen?` check with both?
  It needs to talk to `safe-left?` *AND* `safe-right?`

Have students complete Word Problem: onscreen? (Page 74). When this functions is entered into WeScheme, students should now see that Sam is protected on _both sides of the screen.

Extension Option

What if we wanted to keep Sam safe on the top and bottom edges of the screen as well? What additional functions would we need? What functions would need to change?

Boundary Detection in the Game 10 minutes

Overview

Students identify common patterns between 2-dimensional Boundary detection and detecting whether a player is onscreen. They apply the same problem-solving and narrow mathematical concept from the previous lesson to a more general problem.

Launch

Have students open their in-progress game file and press Run.

- How are the `TARGET` and `DANGER` behaving right now?
  They move across the screen.
- What do we want to change?
  We want them to come back after they leave one side of the screen.
- How do we know when an image has moved off the screen?
  We can see it.
- How can we make the computer understand when an image has moved off the screen?
  *We can teach the computer to compare the image's **coordinates** to a boundary on the number line, just like we did with Sam the Butterfly!*

Investigate

Students apply what they learned from Sam the Butterly to fix the `safe-left?`, `safe-right?`, and `onscreen?` functions in their own code.
Since the screen dimensions for their game are 640x480, just like Sam, they can use their code from Sam as a starting point.

Common Misconceptions

- Students will need to test their code with their images to see if the boundaries are correct for them. Students with large images may need to use slightly wider boundaries, or vice versa for small images. In some cases, students may have to go back and rescale their images if they are too large or too small for the game.

- Students may be surprised that the same code that "traps Sam" also "resets the `DANGER` and \`TARGET\`". It’s critical to explain that these functions do *neither* of those things! All they do is test if a coordinate is within a certain range on the x-axis. There is other code (hidden in the teachpack) that determines *what to do if the coordinate is offscreen*. The ability to re-use function is one of the most powerful features of mathematics - and programming!

Additional Exercises

- Keeping NinjaCat in the Game

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Piecewise Functions

Piecewise Functions

(Also available for Pyret)

Students will learn how one function can have different behaviors based on the input.

Lesson Goals	Students will be able to: Explain what a piecewise function is. Give examples of inputs and outputs of a given ***piecewise function***.
Student-Facing Lesson Goals	I can describe how piecewise functions work.
Materials	Lesson Slides Welcome to Alice's Restaurant! (Page 76) Alice's Restaurant - Explore (Page 77) Alice's Restaurant starter file (Wescheme)

<table>
<tr><td><b>Preparation</b></td><td>Make sure all materials have been gathered
Decide how students will be grouped in pairs</td></tr>
<tr><td><b>Key Points for the Facilitator</b></td><td>The Design Recipe looks a bit different for piecewise, or <i><b>conditional</b></i>, <i><b>function</b></i>s. Check that students are taking time to write <i><b>example</b></i>s and circle what is changing.</td></tr>
<tr><td><b>Language Table</b></td><td>
<table>
<tr><th>Types</th><th>Functions</th><th>Values</th></tr>
<tr><td>Number</td><td><code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>expt</code>, <code>sqr</code>, <code>sqrt</code></td><td><code>4</code>, <code>-1.2</code>, <code>2/3</code>, <code>pi</code></td></tr>
<tr><td>String</td><td><code>string-length</code>, <code>string-repeat</code>, <code>string-contains?</code></td><td><code>"hello"</code>, <code>"91"</code></td></tr>
<tr><td>Boolean</td><td><code><</code>, <code>></code>, <code><></code>, <code><=</code>, <code>>=</code>, <code>string-equal</code>, <code>string<?</code>, <code>string>?</code>, <code>string=?</code>, <code>string<>?</code>, <code>string>=?</code>, <code>and</code>, <code>or</code></td><td><code>true</code>, <code>false</code></td></tr>
<tr><td>Image</td><td><code>star</code>, <code>triangle</code>, <code>circle</code>, <code>square</code>, <code>rectangle</code>, <code>rhombus</code>, <code>ellipse</code>, <code>regular-polygon</code>, <code>radial-star</code>, <code>text</code>, <code>overlay</code>, <code>above</code>, <code>beside</code>, <code>rotate</code>, <code>scale</code>, <code>flip-horizontal</code>, <code>flip-vertical</code></td><td>🔵🔺🔶</td></tr>
</table>
</td></tr>
</table>

Click here to see the prior unit-based version

Glossary

conditional :: a code expression made of questions and answers

example :: shows the use of a function on specific inputs and the computation the function should perform on those inputs

function :: a mathematical object that consumes inputs and produces an output

piecewise function :: a function that computes different expressions based on its input

string :: a data type for any sequence of characters between quotation marks (examples: "hello", "42", "this is a string!")

Not Every Function is Smooth 15 minutes

Overview

Students are challenged via counterexamples to see just how far the Vertical Line Test will go: into behaviors that *feel* like functions but don’t act like a straight line or smooth curve!

Launch

Students should have their computer, workbook, contracts page, and pencil and be logged in to WeScheme and have their workbooks with a pen or pencil.

Have students stand up and put some space between themselves, as if on a number line (each student essentially represents an "x-coordinate"). Give directions to distinct groups of students. For example:

- If you have brown eyes, wave your arms in the air.
- If you have blue eyes, walk in place.
- If you have green or hazel eyes, flap your arms like a chicken.
- If you like sushi, go back to your seat.

Every student should have an activity to perform. Ask a student walking in place why they aren’t waving their arms in the air, or how they knew what to do. Their behavior is essentially the y-coordinate, though for a more direct connection you can specify that different groups sit, kneel, or stand so that their literal *height* represents the y-axis.

> The Vertical Line Test says that to be a function, every input has to be matched with exactly one output.

Ask students: Is this activity representing a function? What is the input? What is the output? *Since each student ("input") has only one action ("output"), it *is still a function** .

Up until now, almost all the functions students have seen are continuous and smooth. Make a big deal about this, so they recognize how big of a shift this is!
Explain that students have just acted out what is called a *piecewise function* . Even though their behavior didn’t follow a smooth pattern (or even a continuous one!), it clearly followed a set of rules and each input had exactly one output. Math has functions like this as well!

Example: Suppose I sell boxes of candy for $2 each. We could imagine that a graph of sales-v-revenue looks like a straight line with a slope of 2: a linear function! But then I want to offer a "bulk discount", where the price drops to $1 for the 21st box of candy and every box after that. Suddenly our line has a kink in it at 20 boxes, where the slope suddenly changes from 2 to 1.
Can students come up with their own examples?

Investigate

Students open the Alice's Restaurant starter file (Wescheme), and turn to Welcome to Alice's Restaurant! (Page 76).
Students investigate the file using their workbook page as a guide.

Notice and Wonder

Have students take time to think and discuss what they Notice and Wonder about this file, which contains some new elements they haven't seen before!

Synthesize

- **What are some familiar things you notice in this file?**

  Answers vary: `define`, `string=?`, a contract and purpose statement, etc.

- **What new things did you notice in this file?**

  Answers vary: the `cond` keyword, the square brackets, `else`, the general look of the `order` function, etc.

- **What function is being defined here? What is its contract?**

  `order`, takes in a String and produces a Number.

- **How do you think this function works?**

  Answers vary - let students drive discussion!

The `order` function is *also* piecewise function! Each input has a single output, but the behavior isn't smooth (there's no relationship between one item's price and another!) or continuous (there are plenty of items not on the menu!).

Partial Functions

For Algebra 2 or pre-calculus teachers, this is a useful time to address *partial functions*. The students who liked sushi had *no rule at all*, meaning that the function was *undefined* at those points. The candy-sales analogy can be extended to say that no one can order more than 100 boxes at a time, making the function undefined for values of x greater than 99.

Defining Piecewise Functions 30 minutes

Overview

Having acted out a piecewise function and examined the code for one on their computers, students take the first step towards writing one, by modifying a function that's already been written for them.

Launch

Students turn to Alice's Restaurant - Explore (Page 77) and complete the exercises with their partner. Students should have added as least one extra option to the menu before moving on.

- **Why do you get an error when you try to use the `sales-tax` function for an item not on the menu?**

  *Let students discuss - move towards the realization that the contract for `order` is `order : String -> Number`, and the "catch-all" branch at the bottom returns a **String** instead of a Number.*

- **What should we do about this?**

  Since we want the program to stop if we give it an invalid input, we should just delete the last branch altogether. Think about other functions that don't work when we give them an invalid input, like dividing by zero!

Investigate

So how do we actually *write* a piecewise function? And more importantly, how does the Design Recipe help us get there?
The Contract and Purpose Statements don't change: we still write down the name, Domain and Range of our function, and we still write down all the information we need in our Purpose Statement (of course, now we might need to write a lot more, since there's more information!).
The examples are also pretty similar: we write the name of the function, followed by some example inputs, and then we write what the function produces with those inputs.

How many examples are needed to fully test this function?

More than two! In fact, we need an example for at least every possible item on the menu!

```
(EXAMPLE (order "hamburger")   6.00)
(EXAMPLE (order "onion rings") 3.50)
(EXAMPLE (order "fried tofu")  5.25)
(EXAMPLE (order "pie")         2.25)
```

Now we circle and label everything that is *change* -able, just as we always have. So what changes?

- The input changes (the String, representing the food being ordered)
- The price changes (the Number, representing the price of the food)

Pedagogy Note

Up until now, there’s been a pattern that students may not have noticed: the number of things in the Domain of a function is *always* equal to the number of labels in the example step, which is *always* equal to the number of variables in the definition. Make sure you explicitly draw students' attention to this here, and point out that this pattern **no longer holds** when it comes to piecewise functions.

> If there are more unique labels in the examples than there are things in the Domain, we’re probably looking at a piecewise function.

We have two things changing (the item and the price), but only one thing is in our Domain. That’s how we know this function is piecewise function!

We start writing the definition as we normally would, using the function name and the input label from the examples step (`(define (order item) …)`). But since we know it’s a piecewise function, now we add `(cond …)` to the body of the function.

Then, for each different behavior we wrote in our examples, we add a condition to the body of our `cond` expression. Each condition has a test and a result, and we copy the results from our examples just as we always do.

```
(define (order item)
  (cond
    [      ...                    6.00]
    [      ...                    3.50]
    [      ...                    5.25]
    [      ...                    2.25]))
```

Finally, we fill in the tests with an expression that tells us *when* the function should behave that way. When should `order` return `6.00`? *when the menu item is "hamburger"!* :

```
(define (order item)
  (cond
    [ (string=? item "hamburger")    6.00]
    [          ...                    3.50]
    [          ...                    5.25]
    [          ...                    2.25]))
```

Extension Activities

Option 1: Students create another function in the code that displays an image of the food instead of the price. This integrates earlier-learned skills in creating images and defining values.

Option 2: Students create a *visual representation* of how the computer moves through a conditional function.

Synthesize

- Can you think of any situations in real life that can be modeled using a piecewise function?
- Is "square root" a piecewise function? Why or Why not?
- Is "absolute value" a piecewise function? Why or Why not?

Player Animation

Player Animation

(Also available for Pyret)

Students apply their knowledge of piecewise functions to write a function to move the player in their game.

Lesson Goals	Students will be able to: Apply previous knowledge of ***piecewise function***s to a new problem situation.
Student-Facing Lesson Goals	I can write a function using conditionals to move my player.
Materials	Lesson Slides Word Problem: update-player (Page 79) Challenges for update-player (Page 80)
Preparation	Make sure all materials have been gathered Decide how students will be grouped in pairs
Key Points for the Facilitator	Encourage students to challenge themselves when creating update-player by completing one of the extension activities.

The update-player function is one of the main places where students can set their game apart and make it theirs. Encourage exploration and experimentation!

Adding comments to code - if you have to ask a student "What are you trying to do there?", then they probably need more comments!

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `>`, `<>`, `<=`, `>=`, `string-equal`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`, `and`, `or`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version

Glossary

contract :: a statement of the name, domain, and range of a function

debug :: to find and fix errors in one’s code

function :: a mathematical object that consumes inputs and produces an output

piecewise function :: a function that computes different expressions based on its input

Defining Piecewise Functions 30 minutes

Overview

Students *define* a piecewise function. This is a challenging task, which is motivated by introducing key events in their video game.

Launch

Students should have their computer, workbook, contracts page, and pencil and be logged in to WeScheme and have their workbooks with a pen or pencil.

You've already defined functions to move your `DANGER` and `TARGET`. Take a moment to look at your code or workbook, and refresh your memory on how they work.

- What controlled the speed of your characters?
- What controlled the *direction* of your characters?

If we wanted our `PLAYER` to go up all the time, we would already know how to do that. If we wanted our `PLAYER` to go *down* all the time, we would already know how to do that. But we want the player to go up *only* when the "up" arrow is pressed, and down when the "down" arrow is pressed. Do we know how to make a function behave differently, based on its input?

Investigate

Students open their **Game Project file** and look for `update-player`, then figure out what the contract represents.

Strategies for English Language Learners

MLR 6 - Three Reads: Have students read through the problem statement three times, looking for different information. What is the problem asking me? What is the ***contract*** for this ***function***? What information do I need to create that function?

- **What is the contract for `update-player`?**

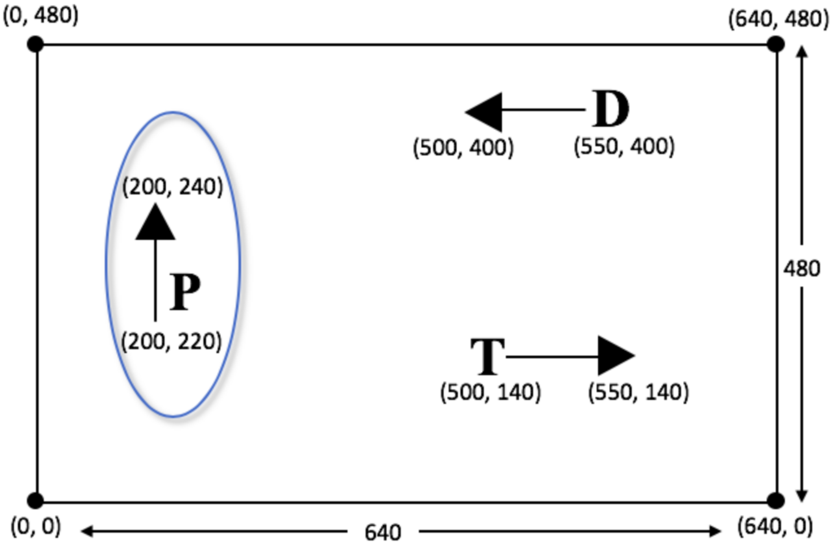
  The Name is `update-player`, *the Domain consists of a Number and String, and Range is a Number.*

- **What does each part of the domain and range represent?**

  Domain: the Number is the y-coordinate of `PLAYER`*, the String is the key that the user pressed; Range: the Number is the new y-coordinate of `PLAYER`*

- **How does the y-coordinate of `PLAYER` change when the user presses the "up" key?**

  It should increase, the program should add something to it



Students complete Word Problem: update-player (Page 79) with a partner, then type their code into their **Game Project file** and test.

Common Misconceptions

- Students often think of this function as returning a *relative distance* (e.g. "it adds 20"), instead of an absolute coordinate (e.g. "the new y-coordinate is the old y plus 20")

Synthesize

- How is this function similar to the piecewise functions you've seen before? How is it different?
- How could we change this function so that the "W" key makes the player go up, instead of the arrow key?
- How could we change this function so that the "W" key makes the player go up, *in addition to* the arrow key?
- Suppose your little brother or sister walks by and hits a random key. What should happen if you hit a random key that doesn't have a meaning in your function? What happens now?

Cheat Codes and Customizations flexible

Overview

Students choose one or more features to make their game more unique. These features can be quite simple, such as adding another key that does the same thing that "up" or "down" does. But they can also be extremely sophisticated, requiring students to exploit properties of the number line in conjuntion with function composition and compound inequalities!

Launch

Right now, all of your games allow the player to move up and down at a constant speed. But what if we wanted to add a special key that made the player warp to the top of the screen, or move down twice as fast? What if we wanted the player to *wrap* , so going off one side of the screen would make it re-appear on the other?

Investigate

Now is your time to customize your game! Try implementing some of the following features, or make your own!

- Warping - program one key to "warp" the player to a set location, such as the center of the screen
- Boundaries - change `update-player` such that `PLAYER` cannot move off the top or bottom of the screen
- Wrapping - add code to `update-player` such that when `PLAYER` moves to the top of the screen, it reappears at the bottom, and vice versa
- Hiding - add a key that will make `PLAYER` seem to disappear, and reappear when the same key is pressed again

Reminder: Use `;` to add comments to code!
Adding useful comments to code is an important part of programming. It lets us leave messages for other programmers, leave notes for ourselves, or "turn off" pieces of code that we don’t want or need to ***debug*** later.
Have students complete at least one of the Challenges for update-player (Page 80) before turning to their computers.

Synthesize

Have students share back what they implemented. Sharing solutions is encouraged!

Question: What would it take to make the player move left and right? Why can't we do this without changing the contract?

ifproglang{wescheme}{ WeScheme supports the ability to change the Domain of a function, which allows `update-player` to take both an x- and a y-coordinate! However, the computer won't know what the new coordinate is if the Range is just a single number. This optional lesson covers the beginnings of *data structures* , teaching just enough to allow students to move their `PLAYER` left and right! }

Pedagogy Note

It's likely that once they hear other students' ideas, they will want more time to try them out. If time allows, give students additional *slices* of "hacking time", bringing them back to share each other's ideas and solutions before sending them off to program some more. This dramatically ramps up the creativity and engagement in the classroom, giving better results than having one long stretch of programming time.

- -

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598).

The Distance Formula

The Distance Formula

(Also available for Pyret)

Students apply their knowledge of the Pythagorean Theorem and Circles of Evaluation to develop a function for the distance formula.

Lesson Goals	Students will be able to: Explain how the distance formula is related to the Pythagorean theorem. Write a function for the distance formula.
Student-Facing Lesson Goals	I can explain how the distance formula is connected to the Pythagorean theorem. I can write a function that takes in 2 points and returns the distance between them.
Materials	Lesson Slides Writing Code to Calculate Missing Lengths (Page 82) Distance on the Coordinate Plane (Page 83) The Distance Between (0, 2) and (4, 5) (Page 84) Multiple Representations: Distance between two points (Page 85) Distance From Game Coordinates (Page 86) Distance (px, py) to (cx, cy) (Page 87) Comparing Code: Finding Missing Distances (Page 88) *Optional: Distance From Game Coordinates 2*
Supplemental Resources	This short video introduces viewers to the nearly 4000 year old Babylonian tablet known as Plimpton 322, which contains a table of Pythagorean Triples that long predates Pythagoras, as well as to Babylonians use of the base 60 system.
Preparation	Make sure all materials have been gathered Decide how students will be grouped in pairs
Key Points for the Facilitator	The distance formula is an excellent review of ***Circles of Evaluation***. Have students work out the expression in small groups to foster discussion.

Language Table

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `>`, `<>`, `<=`, `>=`, `string-equal`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`, `and`, `or`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version

Glossary

circle of evaluation :: a 'sentence diagram' of the structure of a mathematical expression

coordinate :: a number or set of numbers describing an object’s location

interactions area :: the right-most text box in the Editor, where expressions are entered to evaluate

Distance in 1 Dimension 15 minutes

Overview

Students discover the need for distance calculation (first in one dimension, then in two) in video games.

Launch

Students should have their workbook, pencil, and be logged into [WeScheme](https://www.wescheme.org) on their computer.

Open your saved Game File, which should have the Target and Danger moving on their own. Your Player should respond to keypresses, and the Target and Danger should re-appear after they leave the screen. It's almost fully-playable!

- What seems to be missing from this game?

  The characters aren't doing anything when they collide.

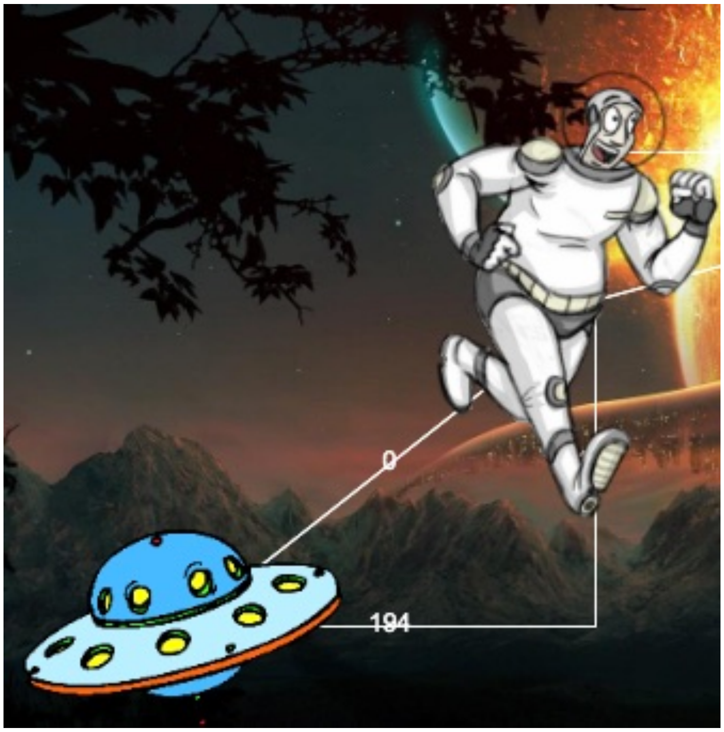
- What does it mean for characters to 'hit' one another? To collide?

  They have to be close enough to touch.

- How will the computer know when the characters have collided?

  When the coordinates of the characters are really close to each other.

Scroll down to the `distances-color` definition (look for `; 4. Collisions` in the file). Right now this value is defined to be the *empty string* `""`. Change this to a color that will show up on your background, and click "Run".



This setting will draw lines from your Player to each of the other characters, and then use those lines as the hypotenuse of right triangles!
The legs of these triangles show the distance in 1 dimension each (on the x- and y-axis). How is this calculated?

Role-Play: Ask a volunteer to help role-play two characters colliding!

- Identify a "number line" on the floor (this can be done just by pointing, or with a visual aid).
- Make sure that you and your volunteer stand with feet as close together as possible, representing the infinitely small point that identifies your center.
- Raise your arms to form a "T shape", representing the outer edges of the characters.
- Emphasize that this represents *one dimension* (perhaps the x- or y-axis).
- With the volunteer, stand about 10 steps away from one another and side-step towards each other one step at a time, while asking the class, "True or False? We are colliding!" *Be sure to only accept "true" and "false" as responses - not "yes" and "no"!*
- Ask the class how far apart you and your volunteer are, and then ask them how they would calculate this if you were standing on a number line and they could see the actual coordinates under your feet.
- After a few iterations, try switching places and repeating. *Point out that students always subtract the smaller number from the larger one, regardless of the character order!*
- Do this until students can clearly see it's when the two characters are 'touching' or 'overlapping' in some way - NOT when they are 'at the same point.'

Investigate

Let's explore how the program computes the length of these lines…

Have students find the `line-length` function in their game files, click Run, and practice using it the ***Interactions area*** with different values for `a` and `b`.

Extension

`line-length` is essentially the way students conceptualize distance in one dimension.
You can extend this `line-length` activity into a lesson on absolute value and have students program `line-length` themselves. Computing 1-dimensional distance - and absolute value - are in fact piecewise functions!

- What does this function *do?*

- Why does it use conditionals?

Synthesize

Make absolutely certain that students understand that this function *always returns the positive distance* between two points on a number line.
What if we have points that are not on the same line? What if instead they differ by both the x- and y-coordinate?

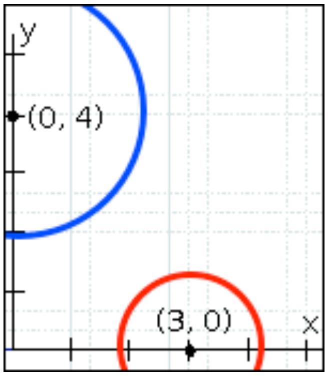
Distance in 2 Dimensions 30 minutes

Overview

Students extend their understanding of *distance* from one dimension to two, using a geometric proof of the Pythagorean Theorem to compute the distance between two points.

Launch

Bring your volunteer (or choose a different one!) back up to the front of the class, and have them squat down on the floor to represent a difference in the y-coordinate between the player and a character. Repeat the role-play activity.



> Suppose the Player is at (0, 4), and another game character is at (3, 0). Now there is a difference in both dimensions. How could we calculate distance *now?*

Computing the distance in 1-dimension is great, as long as the Player and Danger have the same x- or y-coordinate. In that case, the difference between the coordinates is exactly the distance between the two characters. But how do we compute the distance between two points when both the x- *and* y-coordinates are different?

Have students watch video of this problem [Credit: Tova Brown], and try explaining the proof to one another. In our case, the lengths A and B are computed by the `line-length` function we already have!

Have students write code to find the distance between these game characters Writing Code to Calculate Missing Lengths (Page 82).

Why line-length?

Students learn early on that distance in 1-dimension is computed via $|x_2 - x_1|$, and that distance is always a positive value. The Pythagorean Theorem teaches students that the length of the hypotenuse is computed based on the distance in the x- and y-dimension. However, most math textbooks show the distance formula without connecting back to that formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

A student who asks whether it's a problem when $x_2 - x_1$ is negative is displaying a deep understanding of what's going on. Unfortunately, the response to this student relies on a computational artifact of squaring to force a number to be positive (rather than the purpose of squaring in the Pythagorean Theorem). Using the `line-length` function explicitly connects the distance formula back to the 1-dimensional distance students know, allowing them to apply prior knowledge and better connecting back to the Pythagorean Theorem itself. This effectively rewrites the distance formula as:

$$\sqrt{|x_2 - x_1|^2 + |y_2 - y_1|^2}$$

Investigate

Turn to Distance on the Coordinate Plane (Page 83) and look at how line-length is used in the code. See if you can figure out how to write the code for the second problem.

Turn to The Distance Between (0, 2) and (4, 5) (Page 84) in your student workbook. Convert this expression to a Circle of Evaluation, and then to code.

Optional: Have students use this Multiple Representations: Distance between two points (Page 85) to model the distance formula for these coordinates with the Circles of Evaluation.

Practice computing the distance between two *different* points, by completing Distance From Game Coordinates (Page 86).

Optional: Have students complete another pair of these problems using Distance From Game Coordinates 2

Debrief these workbook pages - or have students pair-and-share - before moving on to writing the full distance function.

Using Distance (px, py) to (cx, cy) (Page 87), write a function that takes in two ***coordinate*** pairs (four numbers) of two characters (x_1, y_1) and (x_2, y_2) and returns the distance between those two points. *HINT:* the code you wrote in The Distance Between (0, 2) and (4, 5) (Page 84) can be used to give you your first example!

Students can test their `distance` function using **Pythagorean triples**, such as (3, 4, 5) or (5, 12, 13), to make sure the function is calculating the distance correctly. Finally, students fix the broken `distance` function in their game files. When they click "Run", the right triangles will appear with proper distances for the hypotenuse.

If we knew the lengths of the hypotenuse and one leg of the triangle, could we use the formula $A^2 + B^2 = C^2$ to compute the length of the other leg?

Take a look at the two examples on Comparing Code: Finding Missing Distances (Page 88). There's a subtle difference between the two examples! What is it? Can you explain why they need to be written differently?

Common Misconceptions

It is *extremely common* for students to put variables in the **wrong order**. In other words, their program looks like `…(sqrt (+ (sqr (line-length x1 y1)) (sqr (line-length x2 y2))))…` instead of `…(sqrt (+ (sqr (line-length x2 x1)) (sqr (line-length y2 y1))))…`

In this situation, remind students to look back at what they circled and labeled in the examples step. *This is why we label!*

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.

Collision Detection - Distance and Inequality

Collision Detection - Distance and Inequality

(Also available for Pyret)

Students use function composition and the distance formula to detect when characters in their games collide.

<table>
<tr><td><b>Lesson Goals</b></td><td>Students will be able to:
Explain how the distance formula is related to the Pythagorean theorem.
Write a function for the distance formula.</td></tr>
<tr><td><b>Student-Facing Lesson Goals</b></td><td>I can explain how the distance formula is connected to the Pythagorean theorem.
I can write a function that takes in 2 points and returns the distance between them.</td></tr>
<tr><td><b>Materials</b></td><td>Lesson Slides
Sample game file - no distance lines
Sample game file - with distance lines
Top Down / Bottom Up (Page 89)
Word Problem: collide? (Page 90)
<i>Optional: the Flag of Trinidad and Tobago Starter Code (Wescheme)</i></td></tr>
<tr><td><b>Preparation</b></td><td>Make sure all materials have been gathered
Decide how students will be grouped in pairs</td></tr>
<tr><td><b>Key Points for the Facilitator</b></td><td>The distance formula is an excellent review of <b><i>Circles of Evaluation</i></b>. Have students work out the expression in small groups to foster discussion.</td></tr>
<tr><td><b>Language Table</b></td><td><table><tr><th>Types</th><th>Functions</th><th>Values</th></tr></table></td></tr>
</table>

Number	`+`, `-`, `*`, `/`, `expt`, `sqr`, `sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-length`, `string-repeat`, `string-contains?`	`"hello"`, `"91"`
Boolean	`<`, `>`, `<>`, `<=`, `>=`, `string-equal`, `string<?`, `string>?`, `string=?`, `string<>?`, `string>=?`, `and`, `or`	`true`, `false`
Image	`star`, `triangle`, `circle`, `square`, `rectangle`, `rhombus`, `ellipse`, `regular-polygon`, `radial-star`, `text`, `overlay`, `above`, `beside`, `rotate`, `scale`, `flip-horizontal`, `flip-vertical`	🔵🔺🔶

Click here to see the prior unit-based version

Glossary

circle of evaluation :: a 'sentence diagram' of the structure of a mathematical expression

pixel :: the smallest unit that makes up a digital image. The more pixels, the more detailed an image or video can appear.

Problem Decomposition Returns! 20 minutes

Overview

Students revisit the problem decomposition concept from earlier lessons.

Launch

Students should have their workbook, pencil, and be logged into WeScheme on their computer.
Problem Decomposition is a powerful tool, which lets us break apart complex problems into simpler ones that we can solve, test, and then glue together into a complex solution.
Students may remember that there are two strategies for doing this:

1. **Top-Down:** Describe the problem at a high level, then fill in the details later
2. **Bottom-Up:** Focus on the smaller parts that you're sure of, then build them together to get the big picture

Problem Decomposition is the focus of an entire Bootstrap lesson, is used to solve "onscreen?", and build up the 2-dimensional distance function.

Investigate

For the following complex word problem, have students **first** decide which strategy they want to use, and then apply the Design Recipe to build the functions they need.

> A retractable flag pole starts out 24 inches tall, and can grow at a rate of 0.6in/sec. An elastic is tied to the top of the pole and anchored 200 inches from the base, forming a right triangle. Define functions that compute the height of the pole and the area of the triangle after a given number of seconds.

Have students complete the Top Down / Bottom Up (Page 89) worksheet, using Problem Decomposition and the Design Recipe to solve this problem!

Synthesize

- Which strategy did students use?
- Did they start out with one, and then switch to another?

Collision Detection 20 minutes

Overview

Students once again see function composition at work, as they compose a simple inequality with the `distance` function they've created.

Launch

Knowing how far apart our characters are is the first step. We still need the computer to be asking: "True or False: is there a collision?"

Investigate

Using Word Problem: collide? (Page 90), have students write a function that takes in two coordinate pairs (four numbers) of the `PLAYER` and a character (`(px, py)` and `(cx, cy)`), and and returns `true` if they are within 50 ***pixels*** of each other.

Synthesize

- You started by writing the `distance` function first, and then `collide?` Is this **Top-Down** or **Bottom-Up** decomposition?
- Explicitly point out that this function is easy to write because we can *re-use* the distance function.
- Connect this back to `profit`, `revenue`, `cost` and `onscreen` from previous lessons. Problem Decomposition is powerful!

Additional Exercises:

- For teachers who've already introduced your class to flags, the Flag of Trinidad and Tobago Starter Code (Wescheme) makes use of Pythagorean Theorem and could make for an interesting connection to this lesson.

These materials were developed partly through support of the National Science Foundation, (awards 1042210, 1535276, 1648684, and 1738598). Bootstrap:Algebra by the Bootstrap Community is licensed under a Creative Commons 4.0 Unported License. This license does not grant permission to run training or professional development. Offering training or professional development with materials substantially derived from Bootstrap must be approved in writing by a Bootstrap Director. Permissions beyond the scope of this license, such as to run training, may be available by contacting contact@BootstrapWorld.org.