Name: _____


# Student Workbook


Workbook v3.0

Brought to you by the Bootstrap team:

- Emmanuel Schanzer
- Kathi Fisler
- Shriram Krishnamurthi
- Dorai Sitaram
- Joe Politz
- Jennifer Poole
- Ed Campos
- Ben Lerner
- Flannery Denny

Visual Designer: Colleen Murphy

---

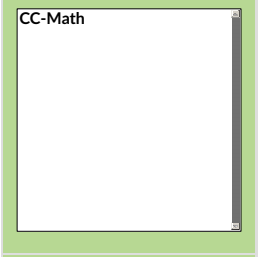
# The Math Inside Video Games

- Video games are all about *change*: How fast is this character moving? How does the score change if the player collects a coin? Where on the screen should we draw a castle?
- We can break down a game into parts, and figure out which parts change and which ones stay the same. For example:
  - Computers use **coordinates** to position a character on the screen. These coordinates specify how far from the left (x-coordinate) and the bottom (y-coordinate) a character should be. Negative values can be used to "hide" a character, by positioning them somewhere off the screen.
  - When a character moves, those coordinates change by some amount. When the score goes up or down, it *also* changes by some amount.
- From the computer's point of view, the whole game is just a bunch of numbers that are changing according to some equations. We might not be able to see those equations, but we can definitely see the effect they have when a character jumps on a mushroom, flies on a dragon, or mines for rocks!
- Modern video games are *incredibly* complex, costing millions of dollars and several years to make, and relying on hundreds of programmers and digital artists to build them. But building even a simple game can give us a good idea of how the complex ones work!

# The Numbers Inside Video Games

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | None |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Identify the objects in a video game that are changing.<br>• Use math language to describe what is changing about each object.<br>• 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.<br>• I can use math vocabulary to describe what is changing about each object.<br>• I understand the time, money, and resources it takes to create a popular video game. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Reverse Engineer worksheet (HTML (Page 3), Google Doc)<br>• NinjaCat demo game (WeScheme) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Key Points for the Facilitator** | • Students will need their own Google accounts.<br>• Take care to manage student expectations about what their game will be like. Modern games are very complex! |
| **Supplemental Resources** | • Coordinates (Quizizz)<br>• The Awesome Coordinate Plane Activity (Desmos Activity)<br>• Boat Coordinate Game (Geogebra)<br>• Coordinate Grid Exploration (Geogebra) |

For a textbook-like version of materials similar to these, you may wish 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.

> **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.

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' Notices.

## *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.)*

## *Possible 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*

- 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.

# Closing

- Share-back: have students share their estimates with the class. Was anything drastically higher or lower than they expected?

# Notice and Wonder

Write down what you notice and wonder about the Ninja Cat game screenshot.

"Notices" should be statements, not questions. What stood out to you? What do you remember?

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

# Reverse Engineer a Video Game

What is changing in the game? The first example is filled in for you.


| Thing in the Game | What Changes About It? | More Specifically? |
|---|---|---|
| Dog | Position | x-coordinate |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Estimating Coordinates


The coordinates for the PLAYER (NinjaCat) are: (_____ , _____)
x y

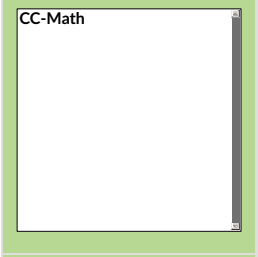
The coordinates for the DANGER (Dog) are: (_____ , _____)
x y

The coordinates for the TARGET (Ruby) are: (_____ , _____)
x y

# 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.

| | |
|---|---|
| **Prerequisites** | None |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Explain the need for ***coordinates*** in a given situation.<br>• Estimate coordinates in a bounded area. |
| **Student-Facing Lesson Goals** | • I can estimate the positions of objects using coordinates.<br>• I can collaborate with a partner to brainstorm a video game.<br>• I can create a sample mock-up (proof of concept) of my video game. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Estimating Coordinates worksheet (HTML (Page 4))<br>• Game Brainstorming organizer (HTML (Page 6))<br>• Optional: cutouts of the Cat, Dog, and Ruby from the NinjaCat game. |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Coordinates (Quizizz)<br>• The Awesome Coordinate Plane Activity (Desmos Activity)<br>• Submarine Coordinate Game (Geogebra)<br>• Coordinate Grid Exploration (Geogebra) |
| **Key Points for the Facilitator** | • The launch activity should create and reinforce the need for coordinates and to attend to precision.<br>• Continue to use the same "Estimating Coordinates" page so students can track their pattern of estimation over time. |

For a textbook-like version of materials similar to these, you may wish 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

---

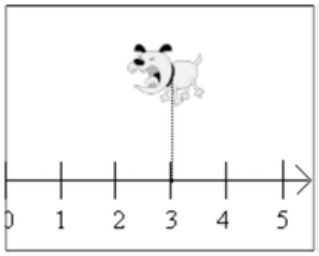
## Navigating a Grid

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.

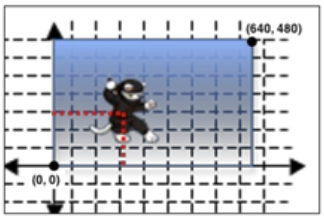
- What is the coordinate of the Dog, if it's on the left-hand edge of the screen?
- What is the coordinate of the Dog, if it's on the right-hand edge of the screen?
- What is the coordinate of the Dog, if it's in the center of the screen?
- What coordinate would place the Dog beyond the left-hand edge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

### Notice and Wonder

As one partner explores the graph, the other student will write down what they Notice on Notice and Wonder (Page 5).
As one partner explores the graph, the other student will write down what they Wonder.

## *Investigate*

- Students complete the Brainstorm Your Own Game 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*

# Notice and Wonder

As one partner explores the Ninja Cat Desmos graph, the other student will write down what they Notice. Students will then switch roles and, as one partner explores the Ninja Cat Desmos graph, the other student will write down what they Wonder.

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

# Brainstorm Your Own Game

Created by: _____

## Background

Our game takes place: _____

In space? The desert? A mall?

## Player

The Player is a _____

The Player moves only up and down.

## Target

Your Player GAINS points when they hit The Target.

The Target is a _____

The Target moves only to the left or right.

## Danger

Your Player LOSES points when they hit The Danger.

The Danger is a _____

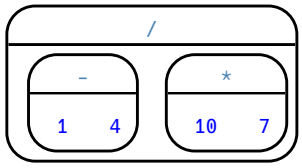
The Danger moves only to the left or right.

## Artwork/Sketches/Proof of Concept

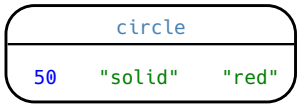
Draw a rectangle representing your game screen, and label the bottom-left corner as the coordinate (0,0). Then label the other four corners. Then, in the rectangle, sketch a picture of your game!

# Starting to Program: Order of Operations & Contracts

- The **Editor** is a software program we use to write Code. Our Editor allows us to experiment with Code on the right-hand side, in the **Interactions Area**. For Code that we want to *keep*, we can put it on the left-hand side in the **Definitions Area**. Clicking the "Run" button causes the computer to read and load everything in the Definitions Area and erase anything that was typed into the Interactions Area.
- Our programming language has many types of **values**:
  - **Numbers** can be integers like `42`, decimals like `0.5`, or even fractions like `1/3`. Clicking on a fraction or a decimal will cause it to switch from one to the other.
  - **Strings** are anything in quotes, such as `"Programming is fun!"`. A Number written in quotes is *still a String!*
- Our language also has **functions** you've seen before, such as addition ( `+` ), subtraction ( `-` ), multiplication ( `*` ) and division ( `/` ).
  - **Order of Operations** is incredibly important when programming. To help us organize our math into something we can trust, we can *diagram* a math expression using the **Circles of Evaluation**. For example, the expression $(1 - 4) \div (10 \times 7)$ can be diagrammed as shown below.




  - To convert a **Circle of Evaluation** into Code, we walk through the circle from outside-in, moving left-to-right. We type an open parenthesis when we *start* a circle, and a close parenthesis when we *end* one. Once we're in a circle, we first write the **function** at the top, then write the inputs from left to right. The circle above, for example, would be programmed as `(/ (- 1 4) (* 10 7))`.
- **Images** are pictures that are produced by functions. The `circle` function, for example, takes a Number as the radius, a String to determine if the circle should be `"solid"` or `"outline"`, and a String to specify the color. You can see the Circle of Evaluation and the Code below:



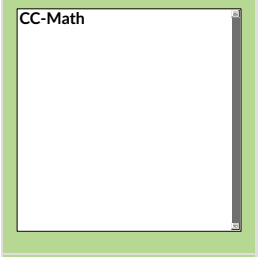

```
(circle 50 "solid" "red")
```

- There are a *lot* of functions in this language! We can make many different shapes, manipulate Strings and Numbers, and a whole lot more. Keeping track of what every function takes in and what it gives back is impossible! To help us remember how to use each function, programmers write down something called a **Contract**. Contracts include the **Name** of the function, what it takes in (called the **Domain**) and what it gives back (called the **Range**). You have space at the very back of your workbook to write all the Contracts for functions that you discover!

# 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".

| | |
|---|---|
| **Prerequisites** | Coordinates and Game Design |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Model an arithmetic expression using ***Circles of Evaluation***.<br>• Translate Circles of Evaluation into code. |
| **Student-facing Goals** | • I can write Circles of Evaluation for a given arithmetic ***expression***.<br>• I can translate a Circle of Evaluation model into code.<br>• I can use numbers and operations in a programming environment. |
| **Materials** | • Lesson slides (Google Slides)<br>• Circles of Evaluation template (Google Doc)<br>• Circles of Evaluation Mixed Review (original (Page 13), solution)<br>• Circles of Evaluation with Square Roots (original (Page 16), solution)<br>• Multiple Representations - Order of Operations (PDF) |
| **Preparation** | • Make sure all materials have been gathered |
| **Supplemental Resources** | • Coordinates, Circles of Evaluation, and Code (Quizizz)<br>• Order of Operations (Quizizz)<br>• Data Types & Circles of Evaluation (Desmos Activity)<br>• Circles of Evaluation Review - Blank Template (Desmos Activity)<br>• Circles of Evaluation Review - Scaffolded (Desmos Activity)<br>• Data Types, Circles of Evaluation, and Contracts (Desmos Activity) |

| Key Points For The Facilitator | • Error messages are the computer trying to give us a clue that something is wrong. Model reacting to ***error message***s with interest to demonstrate to students that the messages are a helpful tool.<br>• 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!<br>• Up until now, we didn't have a visual spatial model for explaining the order of operations. Ask students to compare Circles of Evaluation to previous methods they've learned (PEMDAS, GEMAS, etc)<br>• For a memory hook, model the "bug that crawls through the circle" explanation.<br>• Students may benefit from using multiple colors to distinguish between the different smaller expressions and parentheses. |
|---|---|

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version.

***Glossary***

**circle of evaluation ::** a diagram of the structure of an expression (arithmetic or code)
**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
**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
**interactions area ::** the right-most text box in the Editor, where expressions are entered to evaluate
**value ::** a specific piece of data, like 5 or "hello"

---

# Warmup

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".

---

# Numbers

10 minutes

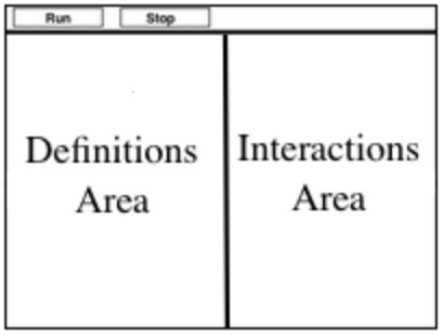
## *Overview*

Students experiment with the Editor, exploring the different kinds of numbers and how they behave in this programming language.

## *Launch*

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. We use nouns, like "bread", "tomato", "mustard" and "cheese" to describe physical objects. Math has ***values***, like the numbers 1, 2 or 3, to describe quantities.

Try typing the number `42` on the right, and then hitting "Enter" or "Return". What did this number evaluate to? (Hint: Numbers should evaluate to themselves - if you didn't get back the same number you put in, something is very wrong!)
If working in pairs, make sure you each take a turn at the keyboard. Suggestions:

- How *large* of a number can you enter?
- How *small* of a number can you enter?
- What happens if you type two numbers on the same line?
- Do fractions work? Decimals?
- Do negative numbers work?

Remember, we're only trying *numbers* for now, not operations like $3 - 6$, $\sqrt{16}$ or $4^2$

### Notice & Wonder

In pairs, students will each try entering a variety of numbers in the Interactions Area, hitting "Enter" each time to see what the computer does. Then they will write down what they Notice and Wonder on Notice and Wonder (Page 8).

- What did you Notice? What do you Wonder?
- Did you get any error messages? If so, read it carefully - what do you think it means?

## *Student Misconceptions*

- Students who try division by writing `3/2` and get an answer may falsely assume that they've performed division. In fact, what they've done is entered a *rational number*. ("Two-thirds" is *equivalent* to the expression "two divided by three", but only insofar as they result in the same value. "2" is equivelent to expression "10 minus 8", for the same reason!)
- Rational numbers can be converted back and forth between fraction and decimal forms by clicking on them.

## *Synthesize*

Our programming language knows about many types of numbers, and they behave pretty much the way they do in math. Our Editor is also pretty smart, and can automatically switch between showing a rational number as a fraction or a decimal, just by clicking on it!

---

# Order of Operations

30 minutes

## *Overview*

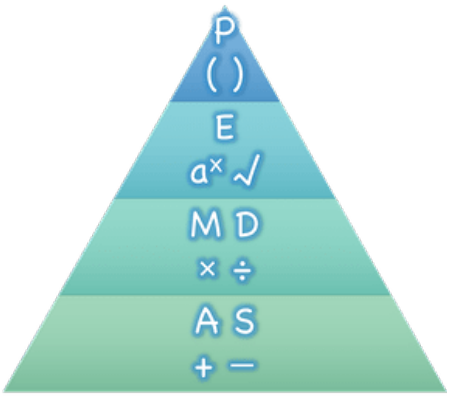
Students are given a challenging expression that exposes common misconceptions about order of operations. The goal is to demonstrate that a brittle, fixed notion of order of operations is *not good enough*, and lead students to a deeper understanding of Order of Operations as a grammatical device. The Circles of Evaluation are introduced as "sentence diagramming for arithmetic".

## *Launch*

Humans also use verbs like "throw", "run", "build" and "jump" to describe operations on these nouns. Mathematics has ***functions*** - or "operations" - like addition and subtraction, which are operations performed on values. Just as you can "**spread** *mustard* on *bread*", a person can also "**add** *four* and *five*".

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

Sometimes, we need multiple expressions to accomplish a task, and it will matter in which order they come. For exmple, if you were to write instructions for making a sandwich, it would matter very much which instruction came first: melting the cheese, slicing the bread, spreading the mustard, etc. The order of functions matters in mathematics, too.




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

But this set of rules is brittle, and doesn't always make it clear what we need to do. 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?

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

Order of Operations mneumonic devices like PEMDAS, GEMDAS, etc focus on how to get the answer. What we need is a *better way to read math*.

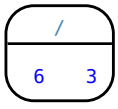
Instead of a rule for computing answers, let's start by diagramming the math itself! We can *draw the structure* of this grammer in mathematics using something called the **Circles of Evaluation**. The rules are simple:

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.

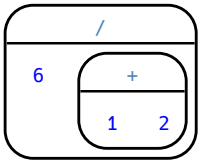
If we want to draw the Circle of Evaluation for $6 \div 3$, the division function ( `/` ) is written at the top, with the `6` on the left and the `3` on the right.



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 divide 6 by that circle?

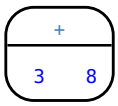
**Circles can contain other Circles**

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



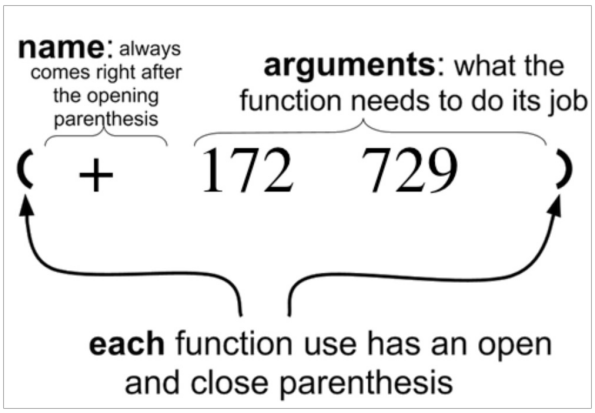

**Circles of Evaluation *help us write code***

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)` |

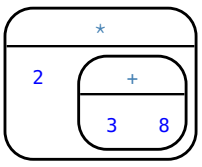


All of the expressions that follow the function name are called arguments to the function. The following diagram summarizes the shape of an expression that uses a function.




Practice creating Circles of Evaluation using the common operators ( `+` , `-` , `*` , `/` ).

- Do spaces matter when typing in functions?
- Does the order of the numbers matter in the functions? Which functions?
- What do the error messages tell us?
- What connections do you see between the expression, circle, and code?

| | | |
|---|---|---|
| **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?

**Circles of Evaluation** ***help us get the correct answer***

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 "any subtraction can be transformed to a negative addition." For example, $1 - 2$ can be transformed to $1 + (-2)$.

Suppose someone tells you that $1 - 2 * 3 + 4$ can be rewritten as $1 + (-2) * 3 + 4$. These two expressions will definitely give us the same answer, but this transformation is actually *incorrect*! It doesn't not use the negative addition rule at all! **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 multiplication happens first, so let's see how multiplication changes this circle:


As you can see, replacing the subtraction with a negative addition happens to the *result* of the multiplication. We can't actually change the `2` into a `-2`, because it isn't actually being subtracted from `1`!

Sure, we got the same answer - but that doesn't mean the way we got it was correct. If all that mattered was getting the right answer, we could just as easily have replaced the whole expression with $5 - 6$. And that is *definitely* not a correct transformation!

Any time you make a transformation in math (replacing $10 - 2$ with $8$ because of subtraction, or replacing $2 + 6$ with $6 + 2$ because of commutativity), you need to make sure the transformation is *correct*. The Circles of Evaluation help us see these transformation *visually*, rather than forcing us to keep them in our heads.

## 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.

### *Investigate*

- Students complete Arithmetic Expressions to Circles of Evaluation & Code (Page 15) page in their workbook. They should *draw all of the Circles first* and check their work, before converting to code.
- Students complete the Translating Circles of Evaluation to Code (Page 13).
- If time allows, partners should take turns entering the code into the editor.

The Circles of Evaluation are a great way to visualize *other* functions you already know, such as square and square root!

**Note:** In WeScheme, we use `sqrt` as the name of the square root function, and `sqr` as the function that squares its input.

- Students complete Translating Circles of Evaluation to Code - w/Square Roots (Page 16) with their partners and test their code in the editor.
- OPTIONAL: Using this graphic organizer, (1) create the code that represents this Circle of Evaluation, (2) translate this into code, (3) evaluate the expression using the order of operations, and (4) then compare and contrast the three methods.

#### 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.

---

# Closing

Have students share back what they learned from the Circles of Evaluation. You may want to assign traditional Order of Operations problems from your math book, but instead of asking them simply to compute the answer - or even list the steps - have them *draw the circle* .

---

# Additional Exercises

- Completing Circles of Evaluation from Math Expressions (1) (original , answers)
- Completing Circles of Evaluation from Math Expressions (2) (original (Page 9) , answers (Page 9))
- Creating Circles of Evaluation from Math Expressions (1) (original , answers)
- Creating Circles of Evaluation from Math Expressions (2) (original , answers)
- Creating Circles of Evaluation from Math Expressions (3) (original (Page 10) , answers (Page 10))
- Converting Circles of Evaluation to Math Expressions (1) (original , answers)
- Converting Circles of Evaluation to Math Expressions (2) (original , answers)
- Matching Circles of Evaluation and Math Expressions (original (Page 11) , answers (Page 11))
- Evaluating Circles of Evaluation (1) (original , answers)
- Evaluating Circles of Evaluation (2) (original , answers)
- Completing Code from Circles of Evaluation (original (Page 12) , answers (Page 12))
- Converting Circles of Evaluation to Code (1) (original , answers)
- Converting Circles of Evaluation to Code (2) (original , answers)
- Matching Circles of Evaluation and Code (original (Page 14) , answers (Page 14))

# Notice and Wonder

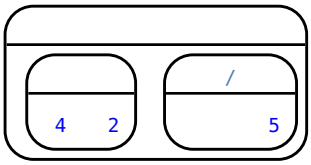
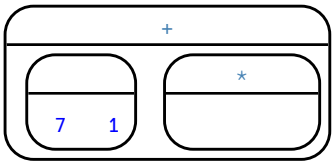
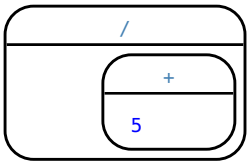
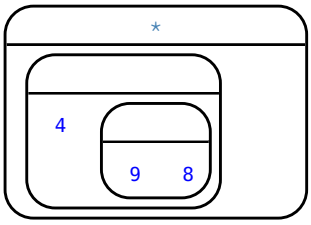
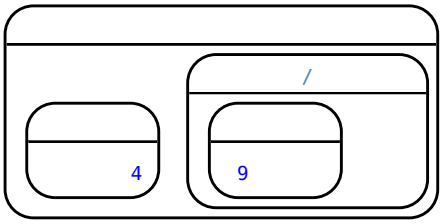
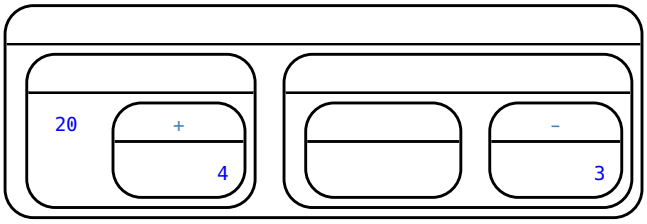
Try typing numbers into the Interactions Area, hitting "Enter", and see what you get back! Some ideas:

1. What is the largest number you can enter? The smallest?
2. Can you write decimals? Fractions?
3. After you get back a decimal, try clicking on it. What happens?
4. Can you write negative numbers? Negative fractions?
5. What else can you try?

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

# Completing Circles of Evaluation from Arithmetic Expressions

For each expression on the left, finish the Circle of Evaluation on the right by filling in the blanks.

| | Arithmetic Expression | Circle of Evaluation |
|---|---|---|
| 1 | $4 + 2 - \frac{10}{5}$ |   |
| 2 | $7 - 1 + 5 \times 8$ |   |
| 3 | $\frac{-15}{5 + -8}$ |   |
| 4 | $(4 + (9 - 8)) \times 5$ |   |
| 5 | $6 \times 4 + \frac{9 - -6}{5}$ |   |
| Challenge | $\frac{20}{6 + 4} - \frac{5 \times 9}{-12 - 3}$ |   |

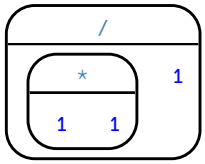
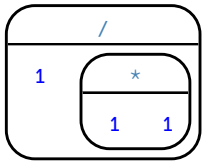
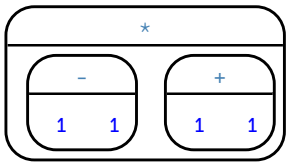
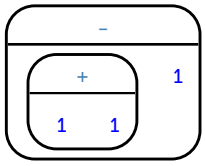
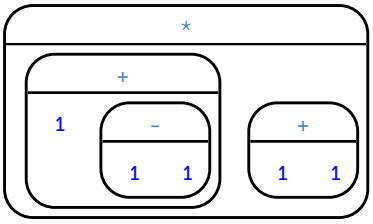
# Creating Circles of Evaluation from Arithmetic Expressions

For each math expression on the left, draw its Circle of Evaluation on the right.

| | Math Expression | Circle of Evaluation |
|---|---|---|
| 1 | $4 - (6 - 17)$ | |
| 2 | $25 + 14 - 12$ | |
| 3 | $1 + 15 \times 5$ | |
| 4 | $\frac{15}{10 + 4 \times -2}$ | |

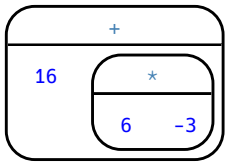
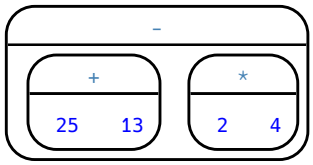
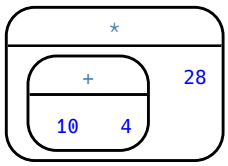
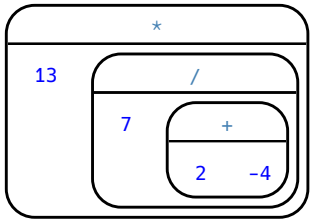
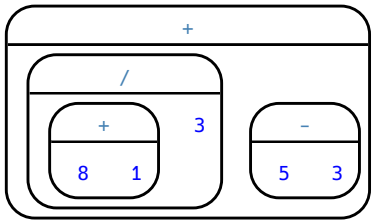
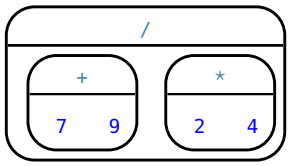
# Matching Circles of Evaluation and Arithmetic Expressions

Draw a line from each Circle of Evaluation on the left to the corresponding arithmetic expression on the right.

| Circle of Evaluation | | | Arithmetic Expression |
|---|---|---|---|
|  | 1 | A | $\frac{1}{1 \times 1}$ |
|  | 2 | B | $1+1-1$ |
|  | 3 | C | $\frac{1 \times 1}{1}$ |
|  | 4 | D | $(1+(1-1)) \times (1+1)$ |
|  | 5 | E | $(1-1) \times (1+1)$ |

# Completing Partial Code from Circles of Evaluation

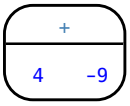
For each Circle of Evaluation on the left, finish the Code on the right by filling in the blanks.

| | Circle of Evaluation | Code |
|---|---|---|
| 1 |  | (+ _____ (* 6 _____)) |
| 2 |  | (_____ (+ _____ 13) (_____ _____ 4)) |
| 3 |  | (_____ (+ _____ 4) _____) |
| 4 |  | (_____ 13 (_____ 7 (_____ 2 -4))) |
| 5 |  | (_____ (_____ (_____ 8 1) 3) (_____ 5 3)) |
| 6 |  | (/ (+ _____ _____) (* _____ _____)) |

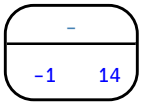
# Translating Circles of Evaluation to Code

Translate the Circles of Evaluation into Code.

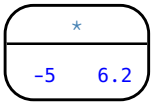
1)



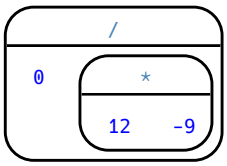

2)



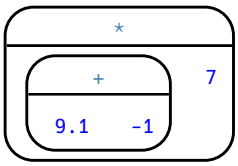

3)



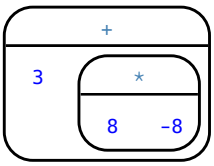

4)



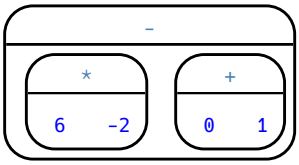

5)



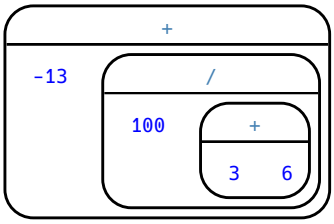

6)



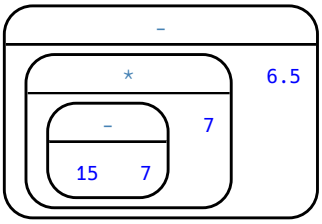

7)




8)

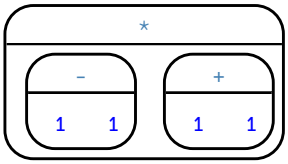
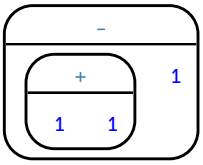
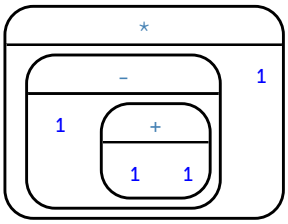
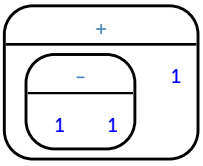
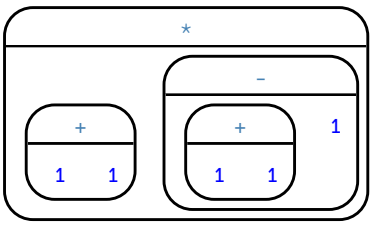



9)



# Matching Circles of Evaluation & Code

Draw a line from each Circle of Evaluation on the left to the corresponding Code on the right.

| Circle of Evaluation | | | Code |
|---|---|---|---|
|  | 1 | A | `(* (- 1 (+ 1 1)) 1)` |
|  | 2 | B | `(* (- 1 1) (+ 1 1))` |
|  | 3 | C | `(* (+ 1 1) (- (+ 1 1) 1))` |
|  | 4 | D | `(- (+ 1 1) 1)` |
|  | 5 | E | `(+ (- 1 1) 1)` |

# Arithmetic Expressions to Circles of Evaluation & Code

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code.

| | Arithmetic | Circle of Evaluation | Code |
|---|---|---|---|
| 1 | $3 \times 7 - (1 + 2)$ | | |
| 2 | $3 - (1 + 2)$ | | |
| 3 | $3 - (1 + 5 \times 6)$ | | |
| 4 | $1 + 5 \times 6 - 3$ | | |

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

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code.

**HINT:** The function name is `sqrt` .

| | Arithmetic | Circle of Evaluation | Code |
|---|---|---|---|
| 1 | $\sqrt{9}$ | | |
| 2 | $\sqrt{5+1}$ | | |
| 3 | $\sqrt{4}+1$ | | |
| 4 | $3\sqrt{3}+\sqrt{7}$ | | |

# Exploring Image Functions

By now you know how to make stars in this programming language. Can you figure out how to make triangles, based on what you know about making stars? Rectangles? What other shapes might we be able to make? When you've discovered code to make a new shape, draw the Circle of Evaluation in the table below, along with a sketch of the shape. Then add the function to your contracts page.

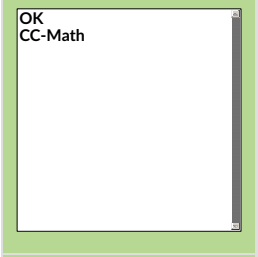
1) Use the space below to draw the Circles of Evaluation for the new functions, and draw a picture of what the function produces.

| Circle of Evaluation | | Image |
|---|---|---|
| star<br>50 "solid" "black" | produces → |  |
| | produces → | |
| | produces → | |
| | produces → | |
| | produces → | |

# Domain and Range

(Also available for Pyret)

Students encounter String and Image datatypes and use "contracts" to make sense of the domain and range of functions.

| | |
|---|---|
| **Prerequisites** | Order of Operations |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Demonstrate understanding of ***Domain*** and ***Range*** and how they relate to ***functions*** |
| **Student-facing Goals** | • I can identify the domain and range of a function.<br>• I can identify the data types ***Number***, ***String***, and ***Image*** |
| **Materials** | • Lesson slides template (Google Slides)<br>• Exploring Image Functions (original (Page 17), solutions)<br>• Reading for Domain and Range (original (Page 18), solution) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Functions Review (Quizizz)<br>• Domain and Range Review (Desmos Activity) |
| **Key Points For The Facilitator** | • Check frequently for understanding of ***data types*** and ***contract***s during this lesson and throughout subsequent lessons.<br>• Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version.

### *Glossary*

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

**datatypes ::** 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

**Image ::** a type of data for pictures

**Number ::** a data type representing a real number

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

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

# Warmup

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".
They will also want to have their Contracts page (back of workbook) ready, preferably in paper form.

# Contracts

15 minutes

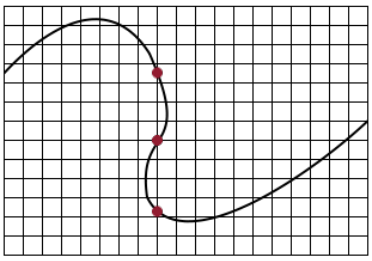
## *Overview*

This activity introduces the notion of ***Contracts***, which are a simple notation for keeping track of the set all of 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!*

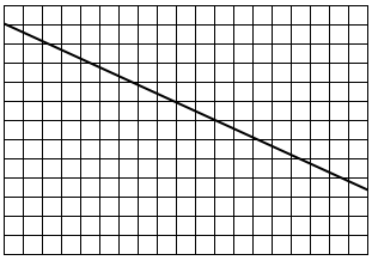
## *Launch*

| For each input to a function, there is exactly one output |
|---|

Functions are a lot like machines: values go in, something happens, and new values come out. Let's start with an example of a function we all know: adding two numbers! Addition is like a machine that takes in pairs of numbers and produces a sum.
Consider the graphs on the right: for every input on the x-axis, a function will produce a *single* output. If we draw a vertical line and it hits the graph more than once, it means there is *more than one output* for the same input. Like any good machine, function machines must be **reliable** .



Not a function



Function

Whenever we use any machine, we always think about what goes in and what comes out. A coffee maker takes in coffee beans and water, and produces coffee. A toaster takes in bread and produces toast. We don't have to know exactly how coffee makers or toasters work in order to *use* them. All we need to know is what type of thing goes in and what type of thing should come out!
In our coffee-maker example, we expect to get the exact same coffee out if we use the exact same beans and water each time. If you put bread in a toaster and got a bagel out, you'd be pretty surprised! *Functions work the same way:* no matter how many times you plug in the same number, you will *always* get the same result. And if you don't? **It's not a function!**

# Investigate

We use something called a ***Contract*** to keep track of what goes in and out of these machines called functions. Contracts are like a "cheat sheet" for using functions. Once you know how to read one, you can quickly figure out how to use a function just by looking at its contract!

> The Contract for a function has three parts: the Name of the function, the ***Domain***, and the ***Range***

- The Name is simply how we refer to the function: `*`, `+`, `sqrt`, etc.
- The ***Domain*** tells us what the function "takes in", or *consumes*. These are also known as the *arguments* to the function.
- The ***Range*** tells us what the function "gives back", *produces*.

Memorizing contracts is hard, and why memorize when we can just keep a log of them! Let's write them down so we can use them later! At the back of your workbook, you'll find pages with space to write down every contract you see in the course.

- What does Multiplication need as an input? What does it produce?
- What inputs does the Square Root function consume? What does it produce?
- When we Square something, what does the Square function consume and produce?
- Write the contracts for `+`, `-`, `*`, `/`, `sqr`, and `sqrt` into the Contracts page.

**A Sample Contracts Table**

| Name | | Domain | | Range |
|---|---|---|---|---|
| `; +` | `:` | `Number Number` | `->` | `Number` |
| `; -` | `:` | `Number Number` | `->` | `Number` |
| `; sqr` | `:` | `Number` | `->` | `Number` |
| `; sqrt` | `:` | `Number` | `->` | `Number` |

It would be silly to buy a coffee-maker that only works with one specific coffee! Similarly, Contracts don't tell us *specific* inputs. They tell us the ***Datatype*** 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 into a mathematical expression.

> Contracts are general. Expressions are specific.

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

## *Synthesize*

- What is wrong with the contract `; + : 3 4 -> 7`?
- What is the difference between a value like `17` and a type like `Number`?

# Exploring Image Functions

25 minutes

## *Overview*

Students explore functions that go beyond numbers, producing all sorts of simple geometric shapes and images in the process. Making images is highly motivating, and encourages students to get better at both reading error messages and persisting in catching bugs.

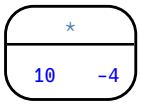
## *Launch*

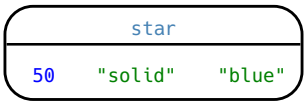
Students have already seen `Number` values like `42`, `-91`, `1/4` or `0.25`, but computer programs can work with a much larger set of ***datatypes***. Show students examples of the `String` datatype, by having them type various things in quotation marks:

- `"hello"`
- `"many words, one string"`
- `"42"`
- `"1/3"`
- Something students come up with on their own…

| A String is *anything* in quotation marks. Like Number values, String values evaluate to themselves. |
|---|

Here are two Circles of Evaluation. One of them is familiar, but the other very different from what you've seen before. What's different about the Circle on the right?




Possible responses:

- We've never seen the function `star` before
- We've never seen Strings 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 Strings

Can you figure out the Name and ***Domain*** for the function in the second Circle? This is a chance to look for and make use of structure in deciphering a novel expression!

Possible responses:

- We know the name of the function is `star`, because that's what is at the top of the circle
- We know it has three things in its Domain
- We know the Domain consists of a Number and two Strings

- But what about the ***Range***? What do you think this expression will evaluate to?
- Convert this Circle to code and try 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.
- What does the `"solid"` mean to the computer? Try replacing it with different values, and see what you get. **If you get an error, read it!** It just might give you a hint about what to do…

You've seen two ***datatypes*** already: Numbers and Strings. Did we get back either on of those? The ***Range*** of `star` is a datatype we haven't seen before: an `Image`!

### 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?

## *Investigate*

- Have students turn to Exploring Image Functions (Page 17) in the workbook.
- Have students open a new program file and name it "Exploring Images".

Give students time to investigate image functions and see how many they can discover, using the Contracts page to organize their findings.

### 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*

- **What image functions did you and your partner discover?** `rectangle`, `triangle`, `ellipse`, `circle`, etc.
- **How did you decide what to try?**
- **What error messages did you see?** *Input mismatches, 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.*

---

# Making Sense of Contracts

10 minutes

## *Overview*

This activity digs deeper into Contracts, and has students create their own Contracts trackers to take ownership of the concept and create an artifact they can refer back to.

## *Launch*

`star` has three elements in its Domain: A Number, a String, and another String.

- **What do these elements represent?** *The Number is the radius, the first String is the style (either* `outline` *or* `solid` *), the second String is the color.*
- **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*
- `square` **has the same Domain as** `star` **. What do the arguments in** `square` **represent?** *length, style, color*
- **Can different functions have the same Domain? The same Range? Are they still different functions?** *Yes, yes, and yes!*
- **Can we come up with an example of two math functions that have the same Domain and Range?**

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

**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.*

## *Investigate*

- Students complete Reading for Domain and Range (Page 18) with their partner.

Students create a visual "Contracts page" either digitally or physically. Ask students to think about how they visualize contracts in their own minds and how they could use that imagery to explain functions and their contracts to others.

---

# Additional Exercises:

- Bootstrap:Algebra - Contracts (Quizizz)
- Bootstrap:Algebra - Data Types & Circles of Evaluation (Desmos Activity)
- Bootstrap:Algebra - Data Types (Desmos Activity)
- Converting Circles of Evaluation to Code (1) (original , answers)
- Converting Circles of Evaluation to Code (2) (original , answers)
- Identifying Parts of Expressions (1) (original , answers)
- Identifying Parts of Expressions (2) (original , answers)
- Matching Expressions & Contracts (original , answers)

# Reading for Domain and Range

As you think about the functions below, remember that you can always type them into your interactions window in the Editor!

| | |
|---|---|
| 1) What is the **name** of the function being used in:<br>`(+ (string-length "broccoli") 8)` | |
| 2) What is the **domain** of the outermost function being used in:<br>`(scale 2 (circle 40 "solid" "blue"))` | |
| 3) What is the **domain** of the innermost function being used in:<br>`(scale 2 (circle 40 "solid" "blue"))` | |
| 4) How many **arguments** does the `+` operator take in:<br>`(+ (string-length "broccoli") 8)` | |
| 5) What is the **range** of the function `string-length` ? | |
| 6) Is `text` a *String* , a *function* , or an *Image* ? | |
| 7) Is the **range** of `text` a *String* or an *Image* ? | |
| 8) What is the first **argument** to the `circle` function in:<br>`(scale 2 (circle 40 "solid" "blue"))` | |

# Composing Image Functions

You'll be investigating these functions with your partner:

```
; text : String Number String -> Image
; scale : Number Image -> Image
; rotate : Number Image -> Image
; flip-horizontal : Image -> Image
; flip-vertical : Image -> Image
```

1) Make an image of your name, in big purple letters. Draw the Circle of Evaluation and write the Code that will create this image.

2) Try using the `scale` function to make your name bigger or smaller. Draw the Circle of Evaluation (hint: use what you wrote above!), then write the Code.

3) In your own words, what does `scale` do?

---

---

4) Try out `rotate`, `flip-horizontal`, and `flip-vertical`. Use the space below to write your Code, then test out your Code in WeScheme when you're ready.

# 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.

| | |
|---|---|
| **Prerequisites** | Domain and Range |
| **Lesson Goals** | Students will be able to:<br>• Demonstrate understanding of the Order of Operations<br>• Use ***Circles of Evaluation*** to combine multiple ***function***s, including non-Number producing functions |
| **Student-facing Goals** | • I can use Circles of Evaluation to combine many kinds of functions. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Function cards (print and cut)<br>• Composing Image Functions (original (Page 19), solutions)<br>• Making Stars (original (Page 20), solutions) |
| **Supplemental Resources** | • Circles of Evaluation Review - Blank Template ( Desmos Activity)<br>• Function Composition Dynamic Illustrator I (Geogebra)<br>• Composition of Functions (Quizizz)<br>• Composition of Function (Geogebra Quiz) |
| **Preparation** | • Make sure all materials have been gathered<br>• 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.<br>• When students encounter errors, encourage them to check their Contracts page and show their work using Circles of Evaluation.<br>• Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version.

***Glossary***

**circle of evaluation ::** a diagram of the structure of an expression (arithmetic or code)
**contract ::** a statement of the name, domain, and range of a function
**datatypes ::** 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
**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

---

# Warmup

Students should be logged into WeScheme and have their workbooks with a pen or pencil.

# Composing Functions

20 minutes

## *Overview*

Students are given a scaffolded activity that forces them to use the output of one function as the input to another - to *compose* them. The Circles of Evaluation are extended to provide a visual-spatial metaphor for function composition, in addition to Order of Operations.

## *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!*

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

## *Investigate*

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.

Have students open to Composing Image Functions (Page 19). Students create a text ***image*** of their name and experiment with their choice of these new functions.

While students are exploring, be available for support but encourage student discussion to solve problems. 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.

Many 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?**

### *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, the percent by which the image should grow or shrink*
- **What does the number in `rotate` represent?** *The rotation angle, measured counterclockwise*
- **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")
```

- **The domain and range for `flip-horizontal` is Image -> Image. Why can I use 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.

---

# Decomposing Image Problems

25 minutes

### *Overview*

Students are given (simple, highly-structured) word problems involving creating images, and must map from the word problems to the names and order of functions needed to solve them. At this stage, the skill is quite brittle and hardly resembles the generalized problem-decomposition skill needed to solve complex word problems in algebra. This is merely the first introduction, and other lessons will deepen and broaden the idea.

### *Launch*

Create the Circles of Evaluation and write the code for the following images. Write a new line of code for each exercise.

- a solid, green `star` of size 50
- a solid, green `star` that is 3 times as large as the original (using the `scale` function)
- a solid, green `star` that is ½ the size of the original (using the `scale` function)
- a solid, green `star` of size 50 that is rotated 45 degrees (using the `rotate` function)
- a solid, green `star` that is 3 times as large as the original and rotated 45 degrees.

## *Investigate*

Students complete Function Composition — Practice (Page 20), practicing drawing Circles of Evaluation and writing code with their partner using different functions.

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.

Create an Image that uses the text function and at least 3 of the following functions:

- `rotate`
- `scale`
- `overlay`
- `flip-horizontal`
- `flip-vertical`
- any other image producing function ( `triangle` , `star` , `circle` , `rectangle` , etc..)

Students should practice writing **comments** in the code to describe what is being produced.

Use `;` at the beginning of a line to write a comment.

---

# Additional Exercises:

- Function Composition Dynamic Illustrator I (Geogebra)
- Composition of Functions (Geogebra Quiz)
- Composite Functions (Quizizz)

# Function Composition — Practice

1) Draw a Circle of Evaluation and write the Code for a **solid, green star, size 50**.

**Circle of Evaluation:**

**Code:** _____

Using the star described above as the **original**, draw the Circles of Evaluation and write the Code for each exercise below.

2) A solid, green star, that is triple the size of the original (using `scale`)

**Circle of Evaluation:**

**Code:** _____

3) A solid, green star, that is half the size of the original (using `scale`)

**Circle of Evaluation:**

**Code:** _____

4) A solid, green star of size 50 that has been rotated 45 degrees counter-clockwise

**Circle of Evaluation:**

**Code:** _____

5) A solid, green star that is 3 times the size of the original **and** has been rotated 45 degrees

**Circle of Evaluation:**

**Code:** _____

# Defining Values and Functions

- We can define values in our program, giving them names that we can refer to later instead of re-typing the same thing over and over. This works the same way it does in math: $x = 5 + 1$ defines the symbol $x$ to be the number $6$.
- In our language, we can define value by writing `(define x (+ 5 1))` . Here are a few value definitions:

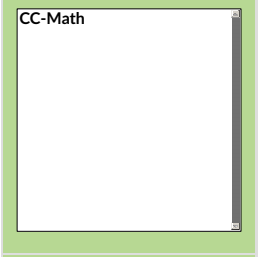
```
(define x (+ 5 1))
(define y (* x 7))
(define food "Pizza!")
(define dot (circle y "solid" "red"))
```

- We can also define new **functions** in our language, to make it do things it didn't do before! To do this, we use a step-by-step process called the **Design Recipe**.
  - The first step is to write the **Contract** for the function you want to build. Remember, a Contract must include the Name, Domain and Range for the function!
  - Then we write a **Purpose Statement**, which is a short note that tells us what the function *should do*. Professional programmers work hard to write good purpose statements, so that other people can understand the code they wrote!
  - The second step is to write at least two **Examples**. These are lines of code that show what the function should do for a *specific* input. Once we see examples of at least two inputs, we can *find a pattern* and see which parts are changing and which parts aren't.
  - Circle the parts that are changing, and label them with a short **variable name** that explains what they do.
  - Finally, the third step is to define the function itself! This is pretty easy after you have some examples to work from: we copy everything that didn't change, and replace the changeable stuff with the variable name!

# Defining Values

(Also available for Pyret)

Students learn how to define lines of code as a set value that can be used repeatedly in different situations, similar to a variable in math.

| | |
|---|---|
| **Prerequisites** | Function Composition |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Demonstrate understanding of ***variable***s and why they are useful in math and programming |
| **Student-facing Goals** | • I can define a line of code as a ***value***, such as `myStar` or `eyeColor`.<br>• I can use my defined value in different situations. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Defining Values Exploration (original (Page 22), solutions)<br>• Defining Values Practice (original (Page 23), solutions) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | |
| **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.<br>• Check frequently for understanding of ***data types*** and ***contract***s during this lesson and throughout subsequent lessons.<br>• Students will use their Contracts page frequently, so it should be kept in an accessible, convenient location. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version.

***Glossary***

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

**datatypes ::** 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, symbol, or term that stands in for a value or expression

---

# Warmup

Students should be logged into WeScheme.

# 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 happened when you were asked to change the color of the star to gold?** *We had 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?

## *Investigate*

You already know how to do this in math: $x = 4 + 2$ evaluates the expression, and *defines* the nickname $x$ to be the value 6.

WeScheme uses the word "define" to make this even clearer! We can type `(define x (+ 4 2))` to define `x` to be the value 6.

- Start a new program, and type this code into the Interactions Area.
- What happens when you hit Enter?
- Can you explain what happened or didn't happen?

| Expressions evaluate to answers. Definitions don't. |
|---|

Think back to math: $x = 4 + 2$ doesn't have an "answer". All it does is tell us that anytime we see $x$, we know it stands for 6. We only see a result when we *use* that definition, for example $x \times 5$ will evaluate to 30.

On the computer, try using the definition of `x` by multiplying it by 5.

- **What is the usefulness of defining values?** *Lets the programmer reuse code, saves time, lets the programmer make changes easily, allows us to more easily use elements inside other functions*
- **What datatypes can we define values for?** *All of them - Number, String, Image*

### 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".

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***.

- Complete Defining Values - Explore (Page 22) in your student workbook. What else can you define?
- Complete Defining Values - Practice (Page 23) with their partner.

---

# Cleaning Up Code

20 minutes

## *Overview*

This activity is a chance to *play* with new concepts, combining value definitions and function composition to create new shapes or to clean up code that generates shapes. The engaging nature of the activity is designed to motivate lots of experiments, each of which gives students a chance to practice applying those concepts.

## *Launch*

The ability to define values allows us to look for - and make use of - structure in our code or in our equations. What structure is repeated in this expression?

$$(x+1)^2 - \frac{4}{(x+1)} * -2(x+1)$$

## *Investigate*

Have students open this file , which draws the Chinese flag.

1. This file uses a function students haven't seen before! What is it?
2. What is it its contract?
3. Have them change the color of all the stars from yellow to black
4. Have them identify what structure is repeated
5. Have them use a value definition to simplify the code
6. Have them change the stars from black back to yellow

**Optional** (for a longer time commitment): Have students choose a flag from this list: (Flags of the World Resource), and recreate one (or more!) of the flags using `define` and any of the other functions they've learned so far.

## *Synthesize*

How many reasons can students come up with for why defining values is useful?

# Defining Values - Explore

```
(define shape1 (triangle 50 "solid" "red"))
```

Type the line of Code above into the Definitions Area of a new program, and press “Run”.

1) What happens when you enter `shape1` into the Interactions Area?

---

---

2) Brainstorm some other values to define. Use the space below to draw any Circles of Evaluation you need and to organize your thoughts.

Ideas: `eye-color` (a String), `age` (a Number), `fav-shape` (an Image)

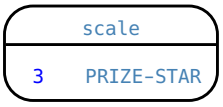
# Defining Values - Practice

1) On the line below, **write the Code** to define `PRIZE-STAR` as the pink outline of a star of size 65.

Using the `PRIZE-STAR` definition from above, draw the Circle of Evaluation and write the Code for each of the exercises. One Circle of Evaluation has been done for you.

2) The outline of a pink star that is 3 times the size of the original (using `scale` )

**Circle of Evaluation:**




**Code:**

3) The outline of a pink star that is half the size of the original (using `scale` )

**Circle of Evaluation:**

**Code:**

4) The outline of a pink star of size 65 that has been rotated 45 degrees

**Circle of Evaluation:**

**Code:**

5) The outline of a pink star that is 3 times the size of the original **and** has been rotated 45 degrees

**Circle of Evaluation:**

**Code:**

6) How does defining values help you as a programmer?

# Notice and Wonder

As you investigate the Game Template file with your partner, record what you Notice, and then what you Wonder.

Remember, "Notices" are statements, not questions.

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

# Making Game Images

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | Defining Values |
| **Relevant Standards** <br> CC-Math | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Apply previous knowledge of ***functions*** to new situations<br>• Use reasoning skills to select appropriate functions and combine their effects |
| **Student-Facing Lesson Goals** | • I can use different functions to transform ***images***.<br>• I can write definitions for my transformed images. |
| **Materials** | • Lesson slides (Google Slides)<br>• Blank Game template (WeScheme)<br>• Linking Images Guide (PDF)<br>• Saving Images Guide (PDF) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | |
| **Key Points for the Facilitator** | • Discuss copyright and fair use guidelines with your students.<br>• Instructional time may vary based on students' experience with using Google Image Search.<br>• Check beforehand for any issues the school Internet security blocker might cause with searching for images.<br>• 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.<br>• Encourage students to focus on finding and scaling each image as needed before moving on to the next one. |

For a textbook-like version of materials similar to these, you may wish 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

# Warmup

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

# The Game Template

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*

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 Template.

### Notice and Wonder

As students investigate the Game Template 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 Template file?**
- **What were some unfamiliar things? Any idea what they might do?** *Answers vary: new functions, comments, images*
- **What datatype is `GAME-TITLE` ? What datatype 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 `bitmap/url` does?**

### What is SCREENSHOT?

The Game Template 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

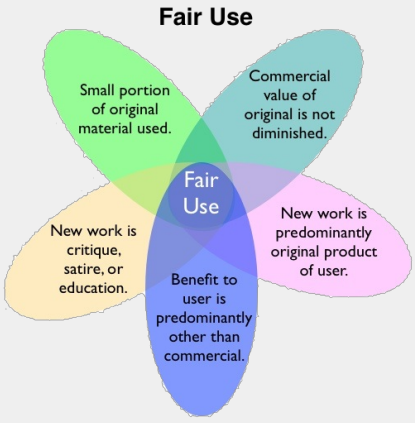
## *Overview*

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

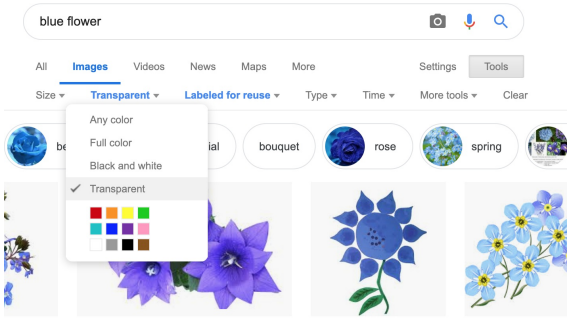
The students will be using images from the Internet for their game, and while this falls entirely under the "Educational Use" umbrella of Fair Use Guidelines, it is still important to make sure students of all ages understand the purpose of copyright law and the differences between educational and commercial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




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.

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:

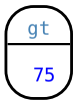
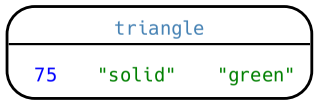
- Save the chosen images to their Drive
- Bring them into the programming environment
- ***Define*** the images as values
- Plan out how to resize and reorient them in their game
- Make sure the final version of each image is defined as either `BACKGROUND`, `TARGET`, `DANGER`, or `PLAYER`

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

# Mapping Examples with Circles of Evaluation

Contract: _____

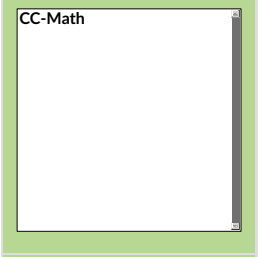
Purpose Statement: _____

| If I type... | → | It should map to... |
|---|---|---|
| EXAMPLE #1: Circle of Evaluation<br>  <br><br>Code: `(gt 75)` | → | Circle of Evaluation:<br>  <br><br>Code: `(triangle 75 "solid" "green")` |
| EXAMPLE #2: Circle of Evaluation<br>Code: | → | Circle of Evaluation:<br>Code: |

# 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.

| | |
|---|---|
| **Prerequisites** | Defining Values |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Describe the usefulness of ***functions***.<br>• Create their own functions and ***examples*** given the constraints of a problem. |
| **Student-Facing Lesson Goals** | • I can explain why a function is useful.<br>• I can plan and create my own function with examples. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Mapping Examples with Circles of Evaluation worksheet (HTML (Page 25))<br>• Fast Functions worksheet (original (Page 26), solutions) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Expression Bundle (Desmos Activities)<br>• Variables and Expressions (Quizizz)<br>• Functions Bundle (Desmos Activities)<br>• Function Notation (Quizizz) |
| **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!<br>• Take plenty of time for the ***Design Recipe*** as students will return to it frequently in future lessons. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

### *Glossary*

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

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

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

**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

**Number ::** a data type representing a real number

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

# Warmup

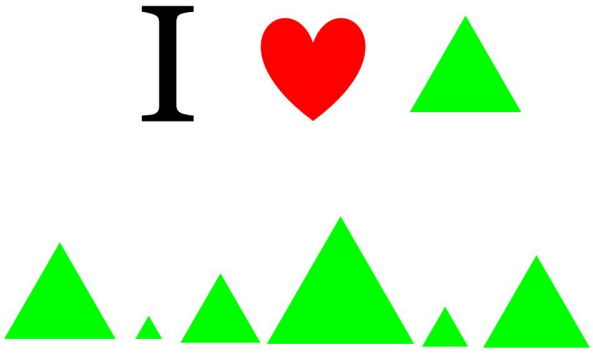
Students should have their workbook, pencil, and be logged into WeScheme on their computer.

# Identifying Repeated *Patterns* 30 minutes

## *Overview*

As with the Defining Values lesson, students search for structure in a list of expressions. But this time, the structures are *dynamic* , meaning they change in a predictable way. This is the foundation for defining functions.

## *Launch*



Confess to your students, "I LOVE green triangles." Challenge them to use the ***Definitions area*** to make as many DIFFERENT solid green triangles as they can in 2 minutes.
Walk around the room and give positive feedback on the green triangles. After the 2 minutes, 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")
```

### Notice and Wonder

Direct students to the various lines of code they came up with. What do you notice? What do you wonder?

- **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 number being given to* `triangle`, *or the* ***Number*** *input.*
- **What strategy did you use to create many different triangles?** *Answers vary: Pattern matching, copy and paste*
- **What shortcut did we use before when we wanted to use the same code over and over?** *We defined values in the Definitions area.*

We've learned how to define **values** when we want to create a shortcut to reuse the same code over and over.
For example: (define myStar (star 50 "solid" "gold"))
**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 called `gt`. When we give it a number, it makes a solid green triangle of whatever size we gave 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. Run through some sample examples before having the class add their own:

- 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")!*

We need to program the computer to be as smart as our volunteer. But how do we do that?

## *Investigate*

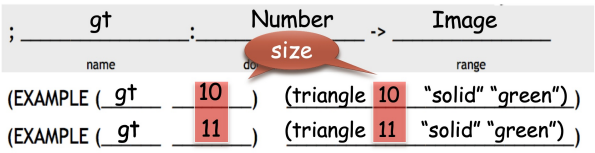
| Word Problem: Write a function called `gt` that takes in a Number and produces a solid, green triangle of that given size. |
|---|

Have students follow along on the Fast Functions (Page 26) handout.

- **1. Write the *contract* for this new function by looking at the word problem.**
  - What does `gt` take in?
    *A Number*
  - What does `gt` give back?
    *An Image. Students may say "a triangle", follow up by asking what data type that triangle will be (Number, String, or Image)*
- **2. Write some examples of how this function should work.**
  - If I typed `(gt 40)` , what would I want the program to do?
    *I'd want the computer the execute the code* `(triangle 40 "solid" "green")` .

*This is a tough question at first. If students are unsure, remind them that we're just writing a shortcut for making green triangles so we don't have to type* `triangle` *, "solid", and "green" every time!*

- OPTIONAL: Have students complete the Mapping Examples with Circles of Evaluation (Page 25) worksheet showing how their function examples are working.
- **3. Circle and Label what is "change-able" - or *variable* between the examples. Circle and label it with a name that describes it.**
  *The number is changing in each example. We could name it "x", but "size" is a more accurate name.*




- **4. Write the function definition.**

*Look at the two examples. The function definition will follow the same pattern, but it will use the variable name* `size` *in place of the variable part we circled. If it doesn't change between the examples, we just copy it.*

```
(define (gt size) (triangle size, "solid", "green"))
```

### 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!

Now that students have defined `gt` have them save their program as 'Defining Functions' and test out their newly created function in the Interactions window.

### *Synthesize*

- **What is the domain for `gt` ?** *Number*
- **Why might someone think the domain for `gt` contains a Number and two Strings, because that's the Domain of `triangle` ?** The function `gt` *uses* `triangle` , but only needs one Number input because *that's the only part that's changing.*
- **Why is defining functions useful to us as programmers?**

---

# Practicing the Design Recipe

flexible

### *Overview*

This is a chance for students to independently review the steps learned in the prior activity, with the teacher in a supporting role asking guiding questions and giving support when needed.

### *Launch*

**Word Problem: Write a function called `gold-star` that takes in number and produces a solid, gold star of that given size.**

- Write 2 examples and the definition of `gold-star` on the 'Fast Functions' handout.
- Complete the `gold-star` example on the Fast Functions (Page 26) worksheet.

### *Investigate*

- Design a problem for a function that takes in one input and returns a shape that uses that input. Your function's input could be a Number, as in the two examples, or a String.
- Write two examples and a definition for your function
- Complete the Mapping Examples with Circles of Evaluation (Page 25) for the examples of your function.

---

# Closing

The Design Recipe is a powerful tool for solving word problems. In this lesson, students practiced using it on simple *programming problems* , but soon they'll be applying it to traditional math problems. Encourage them to make this connection on their own: can they think of a math problem in which this would be useful?

---

# Additional Exercises:

- Review: Define Values & Fast Functions (Desmos Activity)
- Matching Examples & Function Definitions (original , answers)
- Creating Contracts from Examples (1) (original , answers)
- Creating Contracts from Examples (2) (original , answers)

# Fast Functions

There is space below to define four different functions, writing their Contracts, two examples, and the definition itself. The function `gt` - which makes solid green triangles of a given size - is provided as an example. Can you define `bc` as a function which makes solid blue circles of a given *radius*?

```
; gt            :                          Number                          ->     Image

(EXAMPLE (gt              10         ) (triangle  10  "solid"  "green")              )

(EXAMPLE (gt              20         ) (triangle  20  "solid"  "green")              )

(define (gt               size       )

  (triangle  size  "solid"  "green")                                                 )
```

---

```
;              :                                                          ->

(EXAMPLE (                           )                                              )

(EXAMPLE (                           )                                              )

(define (                 )

                                                                                    )
```

---

```
;              :                                                          ->

(EXAMPLE (                           )                                              )

(EXAMPLE (                           )                                              )

(define (                 )

                                                                                    )
```

---

```
;              :                                                          ->

(EXAMPLE (                           )                                              )

(EXAMPLE (                           )                                              )

(define (                 )

                                                                                    )
```

# Word Problem: rocket-height

**Directions**: A rocket blasts off, traveling at 7 meters per second. Use the Design Recipe to write a function `rocket-height`, which takes in a number of seconds and calculates the height.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# 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.

| | |
|---|---|
| **Prerequisites** | Defining Functions |
| **Relevant Standards**<br>OK<br>CC-Math | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Understand how to use the Design Recipe to break down word problems.<br>• Create a strong purpose statement that details in their own words what the function should do. |
| **Student-Facing Lesson Goals** | • I can use the ***Design Recipe*** to break down word problem when writing a ***function***.<br>• I can identify the ***domain*** and ***range*** and other quantities in a word problem when writing a function.<br>• I can create and revise a strong ***purpose statement*** that explains what the function is doing. |
| **Materials** | • Lesson slides (Google Slides)<br>• Rocket-height starter file (WeScheme) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Desmos Expression Bundle (Desmos Activities)<br>• Variables and Expressions (Quizizz)<br>• Desmos Functions Bundle (Desmos Activities)<br>• Functions & Relations (Desmos Polygraph Activity)<br>• Functions Quiz (Quizizz)<br>• Function Notation (Quizizz)<br>• Design Recipe Practice (Desmos Activity)<br>• Design Recipe Practice - Blank Template (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!<br>• Remind students that the domain and range of a function must be one or more of the three ***data types*** (Number, String, Image) they've learned so far.<br>• 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. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

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

**datatypes ::** 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

**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 brief description of what a function does

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

---

# Warmup

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

---

# 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*

Ask students to open the rocket-height Starter File 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.*

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*. 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!

## *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 27) and read the problem statement with their partner.
- Now that the students have revised and refined their purpose statement, have them write the ***Contract*** and ***purpose statement*** on Word Problem: rocket-height (Page 27) worksheet.
- 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 they did with their 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 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:

- Define a function `purple-star`, that takes in the size of the star and produces an outlined, purple star of the given size.
  Word Problem: purple-star
- Define a function `spot`, that takes in a color and produces a solid circle of radius 50, filled in with that color. (Word Problem: spot)
- Define a function `average`, which takes in two numbers and produces their average. (Word Problem: average)
- Do Examples Have the Same Contracts? (1) (original, answers)
- Do Examples Have the Same Contracts? (2) (original, answers)
- Matching Contracts and Examples (1) (original, answers)
- Matching Contracts and Examples (2) (original, answers)

# Writing Quality Purpose Statements

## 3 Reads

| 1st Read: What is this problem about? | 2nd Read: What are the Quantities? |
| --- | --- |
| 3rd Read: What is a good Purpose Statement? | |

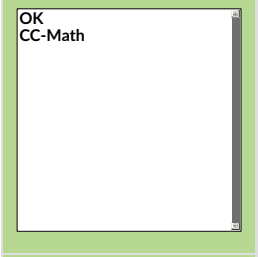
## Stronger & Clearer

| Purpose Statement 1st Revision: |
| --- |
| Purpose Statement 2nd Revision: |

# 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.

| | |
|---|---|
| **Prerequisites** | Defining Functions |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Understand how to use the ***Design Recipe*** to break down simple word problems.<br>• 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.<br>• I can identify ***domain*** and ***range*** and other quantities in a word problem when writing a function.<br>• I can create and revise a strong purpose statement that explains what the function is doing. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Purpose Statement organizer (HTML (Page 28))<br>• Word Problems 1 (HTML (Page 30))<br>• Word Problems 2 (HTML (Page 31))<br>• Word Problems 3 (HTML (Page 32))<br>• Word Problems 4 (HTML (Page 33))<br>• Word Problems 5 (HTML (Page 34))<br>• Word Problems 6 (HTML (Page 35)) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |

| | |
|---|---|
| **Supplemental Resources** | • Expression Bundle (Desmos Activities)<br>• Desmos Modeling Bundle (Desmos Activities)<br>• Functions Bundle (Desmos Activities)<br>• Functions & Relations (Desmos Polygraph Activity)<br>• Linear Bundle (Desmos Activities)<br>• Quadratics Bundle (Desmos Activities)<br>• Exponential Bundle (Desmos Activities)<br>• Linear, Quadratic, and Exponential Equations (Quizizz)<br>• Variables and Expressions (Quizizz)<br>• Functions Quiz (Quizizz)<br>• Function Notation (Quizizz)<br>• Linear Equations (Quizizz)<br>• Quadratic Equations (Quizizz) |
| **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!<br>• The domain and range of a function are described as ***data types***, such as Number, String, or Image.<br>• If students struggle with getting started, encourage them to start with one example and use the Circles of Evaluation ***example***s mapping organizer.<br>• This activity can work well as a formative review.<br>• This activity is a good time to get students working with someone other than their usual coding partner. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

**datatypes ::** 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
**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
**purpose statement ::** a brief description of what a function does
**range ::** the type or set of outputs that a function produces

---

# Warmup

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

---

# 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 will use the Purpose Statement organizer (Page 28) 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.
- 1st new partner: Read their purpose statements to each other & revise the purpose statement to be stronger and clearer.
- 2nd new partner: Repeat.
- 3rd new partner: Repeat.

Students may choose to use the programming environment to test out their functions or to solve solution disputes. 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, but you can also grab any word problem from your math book in which students must define a functional relationship.

**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*

Strong 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").

# Additional Exercises:

- Bootstrap Algebra: Design Recipe (Desmos Activity)
- Bootstrap Algebra: Design Recipe Practice (Blank Template) (Desmos Activity)
- Bootstrap: Algebra - More Design Recipe Practice (Desmos Activity)
- Bootstrap: Algebra - Coordinates (Quizizz)

# Mapping Examples with Circles of Evaluation

Contract: _____

Purpose Statement: _____

| If I type… | → | It should map to… |
|---|---|---|
| EXAMPLE #1: Circle of Evaluation<br><br>Code: | → | Circle of Evaluation:<br><br>Code: |
| EXAMPLE #2: Circle of Evaluation<br><br>Code: | → | Circle of Evaluation:<br><br>Code: |

# The Design Recipe

**Directions**: Write a function `marquee` that takes in a message and returns that message in large gold letters.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

---

**Directions**: Write a function `circle-area` that takes in a radius and returns the area of the circle.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# The Design Recipe

**Directions**: Write a function `minimum-wage`, that takes in a number of hours worked and returns the amount a worker will get paid at $10.25/hr.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Write a function `tip-calculator` that takes in the cost of a meal and returns the 15% tip for that meal.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# The Design Recipe

**Directions**: Getting a gym membership costs $150, and then there's a $45/month fee after that. Write a function `globo-gym` that takes in a number of months and produces the cost of a membership for that many months.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: The cost of a ride is a starting price of $2.50, plus $1.50/mile. Write a function `rideshare`, that takes in a number of miles and produces the cost of that right.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# The Design Recipe

**Directions**: Write a function `moving` that takes in the days and number of miles driven and returns the cost of renting a truck. The truck is $55 per day and each driven mile is 15¢.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Write a function `lawn-area` that takes in the length and width of a rectangular lawn and returns its area.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# The Design Recipe

**Directions**: Write a function `rect-perimeter` that takes in the length and width of a rectangle and returns the perimeter of that rectangle.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Write a function `rectprism-vol` that takes in the length, width, and height of a rectangular prism and returns the Volume of a rectangular prism.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# The Design Recipe

**Directions**: Write a function `split-tab` that takes in a cost and the number of people sharing the bill and splits the cost equally.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Write a function `num-cube` that takes in a number and returns the cube of that number.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# Danger and Target Movement

**Directions**: Use the Design Recipe to write a function `update-danger`, which takes in the danger's x-coordinate and produces the next x-coordinate.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Use the Design Recipe to write a function `update-target`, which takes in the danger's x-coordinate and produces the next x-coordinate.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

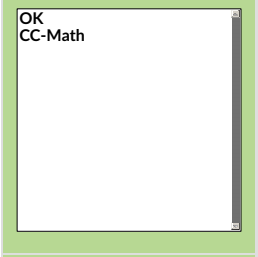
(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# Character Animation

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | Restating the Problem |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Apply the ***Design Recipe*** to create a ***function*** given the constraints of a word problem.<br>• Explain the basics of animation. |
| **Student-Facing Lesson Goals** | • I can use the Design Recipe to make a function.<br>• I can describe how animation works. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Danger and Target Movement (Page 36) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | |
| **Key Points for the Facilitator** | • Encourage students to take their time in understanding *why* we want to fix `update-danger` and `update-target`.<br>• Students might be confused as to *how* the animation is working. The `make-game` function at the bottom of the file has many inputs - including `update-danger` and `update-target`. `make-game` 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! |

For a textbook-like version of materials similar to these, you may wish 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

---

# Warmup

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

---

# Animation

45 minutes

## *Overview*

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

## *Launch*

- How does animation work?
- 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-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 Template 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 36), and transfer the code to their Game Template 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 36), with their partner and transfer the code to their Game Template 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!

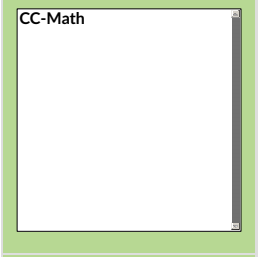
# Problem Decomposition

- Sometimes a problem is too complicated to solve all at once. Maybe there are too many variables, or there is just so much information that we can't get a handle on it!
- We can use **Problem Decomposition** to break those problems down into simpler pieces, and then work with the pieces to solve the whole. There are two strategies we can use for decomposition:
  - **Top-Down** - Start with the "big picture", writing functions or equations that describe the connections between parts of the problem. Then, work on defining those parts.
  - **Bottom-Up** - Start with the smaller parts, writing functions or equations that describe the parts we understand. Then, connect those parts together to solve the whole problem.
- You may find that one strategy works better for some types of problems than another, so make sure you're comfortable using either one!

# 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.

| | |
|---|---|
| **Prerequisites** | Restating the Problem |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Write a ***function*** that explicitly uses another function.<br>• Explain the benefits and drawbacks of functions that depend on each other.<br>• 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.<br>• I can write a function that uses another function. |
| **Materials** | • Lesson slides (Google Slides)<br>• Design Recipe: revenue (PDF (Page 38))<br>• Design Recipe: cost (PDF (Page 39))<br>• Design Recipe: profit (PDF (Page 40))<br>Bootstrap Formative Assessments<br>• Bootstrap: Algebra - Coordinates, Circles of Evaluation, & Code (Quizizz)<br>• Bootstrap:Algebra - Data Types & Circles of Evaluation (Desmos Activity)<br>• Bootstrap:Algebra - Circles of Evaluation Review(Blank Template) (Desmos Activity)<br>• Bootstrap:Algebra - Contracts, Domain/Range, Data Types, & Functions (Quizizz)<br>• Bootstrap:Algebra - Data Types, Circles of Evaluation, and Contracts (Desmos Activity) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Function Composition Dynamic Illustrator I (Geogebra )<br>• Composition of Functions (Geogebra Quiz )<br>• Composition of Functions 2 (Quizizz ) |
| **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.<br>• 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?<br>• 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. |

***Glossary***

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

---

# Warmup

Students should have their workbook, pencil, and be logged into WeScheme and have their workbooks with a pen or pencil.

---

# Problem Decomposition

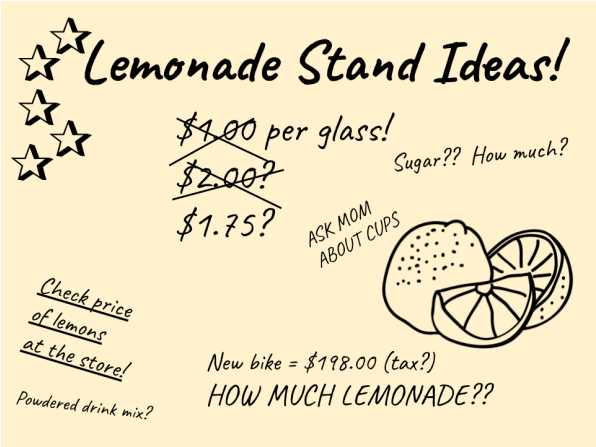
30 minutes

## *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*

Display the following image:




> **Notice and Wonder**
>
> What do you notice? What do you 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 38), cost (Page 39), and profit (Page 40), 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. |
|---|

When thinking Bottom-Up, we start with the small, easier relationships first and then build our way to the larger relationships. In the Lemond Stand example, we had you write the lower-level functions - `cost` and `revenue` - *first*, and then gave you the chance to use them in the higher-level `profit` function. This is called **Bottom-Up** design.
When thinking Top-Down, we start with the "big picture" and then worry about the details later. For example, we could have *started* with `profit`, and kept track of all the lower-level functions we would need to write. This is called **Top-Down** design.

## *Investigate*

Consider the following situation:
*Jamal's trip to Thailand requires him to drive 20 miles to the airport, take a plane 9,000 miles to Thailand, and then a bus 6 miles to his hotel. The average speed when driving to the airport is 40mph, the average speed of an airplane is 575mph, and the average speed of his shuttle bus is 15mph*
**Setting aside time spent waiting at the airport or for the bus, how long is Jamal in transit?**
This problem can be decomposed in Top-Down or Bottom-Up fashion. Describe what your steps would be in each solution (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.

# Word Problem: revenue

**Directions**: Use the Design Recipe to write a function `revenue`, which takes in the number of glasses sold at $1.75 apiece and calculates the total revenue.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Word Problem: cost

**Directions**: Use the Design Recipe to write a function `cost`, which takes in the number of glasses sold and calculates the total cost of materials if each glass costs $.30 to make.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Word Problem: profit

**Directions**: Use the Design Recipe to write a function `profit` that calculates total profit from glasses sold, which is computed by subtracting the total cost from the total revenue.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

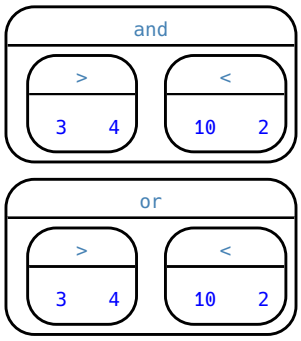
*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Inequalities

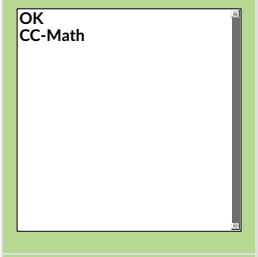
- Sometimes we want to *ask questions* about data. For example, is `x` greater than `y`? Is one string equal to another? These questions can't be answered with a Numbers. Instead, they are answered with a new datatype called a **Boolean**.
- Video games use Booleans for many things: asking when a player's health is equal to zero, whether two characters are close enough to bump into one another, or if a character's coordinates put it off the edge of the screen.
- A Boolean value is either `true` or `false`. Unlike Numbers, Strings, and Images, Booleans have only two possible values.
- You already know some functions that produce Booleans, such as `<` and `>`! Our programming language has them, too: `(< 3 4)`, `(> 10 2)`, and `(= -10 19)`.
- We also have ways of writing **Compound Inequalities**, so we can ask more complicated questions using the `and` and `or` functions.
  - `(and (> 3 4) (< 10 2))` translates to "three is less than four *and* ten is less than two". This will evaluate to `false`, since the `and` function requires that both sub-expressions be `true`.
  - `(or (> 3 4) (< 10 2))`, which translates to "three is less than four *or* ten is less than two". This will evaluate to `true`, since the `or` function only requires that one sub-expression be `true`.
- The Circles of Evaluation work the same way with Booleans that they do with Numbers, Strings and Images:



# Simple Inequalities

(Also available for Pyret)

Students discover the Boolean data type, and apply knowledge of inequalities to simple programming problems.

| | |
|---|---|
| **Prerequisites** | Problem Decomposition |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Describe the solution set of a simple inequality<br>• Explain the 'Boolean' datatype |
| **Student-Facing Lesson Goals** | • I can use two or more inequalities together and describe the area they enclose.<br>• I can explain to someone else what a Boolean is. |
| **Materials** | • Lesson slides (Google Slides)<br>• Sam the Butterfly Starter File (WeScheme)<br>• Inequalities Launch worksheet (original (Page 42), solutions)<br>• Inequalities Explore worksheet (original (Page 43), solutions)<br>• Left-and-Right (original (Page 44), solution) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Booleans Review (Quizizz)<br>• Booleans Activity (Desmos Activity)<br>• Inequalities Bundle (Desmos Activities)<br>• Inequalities & Graphing Inequalities (Quizizz)<br>• Inequality Graph Illustrator (Geogebra) |
| **Key Points for the Facilitator** | • A ***Boolean*** is just another ***data type***, like Number, or Image, but unlike the others there are only two values: `true` and `false`. While simple to explain, this different behavior can be confusing for some students.<br>• Functions that produce Booleans are typically questions, so the names of the functions in this lesson read like questions.<br>For example, `safe-left?`, `onscreen?` are both functions that are asking if a condition, such as an image being on the screen, is true or false. * Role-playing can help students understand the jobs of `safe-left?` and `safe-right?`. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

**Boolean ::** a type of data with two values: true and false
**coordinate ::** a number or set of numbers describing an object's location
**datatypes ::** a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure
**expression ::** a computation written in the rules of some language (such as arithmetic, code, or a Circle of Evaluation)

---

# Warmup

Students should have their workbook, pencil, and be logged into WeScheme and have their workbooks with a pen or pencil.

---

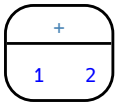
# Introducing Booleans 15 minutes

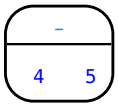
## *Overview*

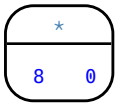
Students discover the concept of inequalities (or apply it, if they've seen it before) in programming, and extend their knowledge of data types, Contracts, and Circles of Evaluation.

## *Launch*

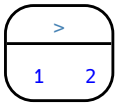
Ask students to evaluate Circles of Evaluation for simple expressions they've seen before, and ask them to convert them into code.

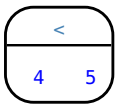
- 

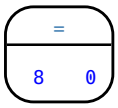
- 

- 


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

- 

- 

- 


Have student type in these expressions. What did they get?

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 datatype* , called a ***Boolean***. Booleans are the answers to any yes-or-no question (for example: "Is five greater than two?", "Did a character hit a wall?", etc.)

## *Investigate*

Have students open to the Inequalities — Launch (Page 42) worksheet and complete with a partner.

## *Synthesize*

- Students will see functions on the worksheet that they've never encountered before! But instead of answering their questions, encourage them to make a *guess* about 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.

---

# 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 and click "Save."

Have students turn to the Introducing Sam (Page 43) and investigate the program with their partner.

*Let students figure out that they need to press "Run" to see what the program does, and that the arrow keys control Sam.*

- **What is something you noticed about this program?** *Sam can be moved with the arrow keys, 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 490*
- **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 44). 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.

## *Synthesize*

- 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.
- For each of them, 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)!") **Note:** Do not ask `onscreen?` to roleplay beyond their contract! They'll get involved in the next lesson...

## *Additional Exercises*

- Keeping Ninjacat in the Game (original , answers)
- Converting Circles of Evaluation with Booleans to Code (original , answers)
- Converting Circles of Evaluation with Booleans to Code (original , answers)

# Inequalities—Launch

What would each of the following expressions evaluate to? Write your guesses in the space provided, and then take turns typing them into the computer.

| | |
|---|---|
| 1) `(+ 1 4)` will be _____ | 2) `(> 0 5)` will be _____ |
| 3) `(/ 4 2)` will be _____ | 4) `(= 1 9)` will be _____ |
| 5) `(- 0 9)` will be _____ | 6) `(<= 2 2)` will be _____ |

8) What does the function `<` do?

---

9) What does the function `(string=?)` do?

---

10) Write the contract(s) for any new function(s) that produce Booleans you've seen above in your Contracts page.

11) How many Numbers are there in the entire universe? _____

12) How many Strings are there in the entire universe? _____

13) How many Images are there in the entire universe? _____

14) How many Booleans are there in the entire universe? _____

What are they?

---

# Sam the Butterfly

Open the “Sam the Butterfly” starter file and press “Run”. Hi, Sam!
Move Sam around the screen using the arrow keys.

1) What changes as the butterfly moves left and right? _____

Sam is in a 640 × 480 yard. Sam’s mom wants Sam to stay in sight.

**How far to the left and right can Sam go and still remain visible?**

Use the new inequality functions to answer the following questions *with code*:

2) Sam hasn’t gone off the left edge of the screen as long as… _____

2) Sam hasn’t gone off the right edge of the screen as long as… _____

4) Use the space below to draw Circles of Evaluation for these two expressions:

# Left and Right

**Directions**: Use the Design Recipe to write a function `safe-left?`, which takes in an x-coordinate and checks to see if it is greater than -50.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

---

**Directions**: Use the Design Recipe to write a function `safe-right?`, which takes in an x-coordinate and checks to see if it is less than 690.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____
*function name* *domain* *range*

; _____
*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )
*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )
*function name* *variable(s)*

_____ )
*what the function does with those variable(s)*

# Inequalities—Practice

Create the Circles of Evaluation, then convert the expressions into Code in the space provided.

1) 2 is less than 5, and 0 is equal to 6

What will this evaluate to? _____

2) 6 is greater than 8, or -4 is less than 1

What will this evaluate to? _____

3) The String “purple” is the same as the String “blue”, and 3 plus 5 equals 8

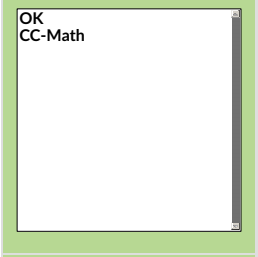
What will this evaluate to? _____

4) Write the contracts for `and` & `or` in your Contracts page.

# Compound Inequalities

(Also available for Pyret)

Students learn to compose inequalities using the concepts of union and intersection, and solve problems using compound inequalities. Finally, they apply what they've learned to set screen boundaries in their game.

| | |
|---|---|
| **Prerequisites** | Simple Inequalities |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Describe how functions can work together.<br>• Describe the solution set of a compound inequality<br>• Make mathematical adjustments relevant to their game. |
| **Student-Facing Lesson Goals** | • I can use two or more inequalities together and describe the area they enclose.<br>• I can tell someone else how two or more ***function***s work together.<br>• I can make adjustments to a program based on how the program behaves. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Inequalities Explore worksheet (original (Page 45), solutions)<br>• Design Recipe: onscreen? (original (Page 46), solution)<br>Bootstrap Formative Assessments<br>• Booleans Review (Quizizz, Desmos Activity) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Desmos Inequalities Bundle (Desmos Activities)<br>• Inequalities & Graphing Inequalities (Quizizz)<br>• Inequality Graph Illustrator (Geogebra)<br>• Graphing Compound Inequalities (Quizizz) |
| **Key Points for the Facilitator** | • Role-playing can help students understand the job of `onscreen` ?, and how it relates to `safe-left?` and `safe-right?` .<br>• If a student's `TARGET` and `DANGER` image seem to be "getting stuck" on the edge of the screen, the student may have to adjust the side boundaries depending on the size of their images.<br>• The code for the boundary functions in the game is *exactly the same* as in Sam the Butterfly. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

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

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

# Warmup

Students should have their computer, contracts page, and pencil and be logged in to WeScheme with their Game Project file open.

# Compound Inequalities 10 minutes

## *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? ($temperature > 80^\circ$)
- Did I get paid enough for painting that fence? ($temperature < \$100$)
- Are the cookies finished baking? ($timer = 0$)

Have students come up with other examples.

But many times we need to *combine* inequalities:

- Should I go to the beach? ($temperature > 80^\circ$ and $weather =$" $sunny$ ")
- Was this burrito worth the price? ($taste =$" $delicious$ " and $price \leq \$20$)

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.

Have students complete Inequalities — Practice (Page 45).

## *Synthesize*

- Be really careful to check for understanding here. *Expresions using* `and` *only produce* `true` *if both of their sub-expressions are* `true`. *Expresions 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!

# 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 46). 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?}

---

# Boundry 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 numeric boundary, 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:

- Word Problem: hot?
- Word Problem: sunny?
- Word Problem: beach-day?

# Word Problem: onscreen?

**Directions**: Use the Design Recipe to write a function `onscreen?`, which takes in an x-coordinate and checks to see if Sam is safe on the left while also being safe on the right.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Piecewise Functions

- Sometimes we want to build functions that act differently for different inputs. For example, suppose a business charges $10/pizza, but only $5 for orders of six or more. How could we write a function that computes the total price based on the number of pizzas?
- In math, **Piecewise Functions** are functions that can behave one way for part of their Domain, and another way for a different part. In our pizza example, our function would act like $cost(pizzas) = 10 * pizzas$ for anywhere from 1-5 pizzas. But after 5, it acts like $cost(pizzas) = 5 * pizzas$.
- Piecewise functions are divided into "pieces". Each piece is divided into two parts:
  1. How the function should behave
  2. The domain where it behaves that way
- Our programming language can be used to write piecewise functions, too! Just as in math, each piece has two parts:

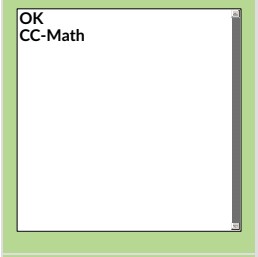
```
(define (cost pizzas)
  (cond
    [(< pizzas 6) (* 10 pizzas)]
    [(>= pizzas 6) (* 5 pizzas)]))
```

- Piecewise functions are powerful, and let us solve more complex problems. We can use piecewise functions in a video game to add or subtract from a character's x-coordinate, moving it left or right depending on which key was pressed.

# Piecewise Functions

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | Simple Inequalities |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Explain what a piecewise function is.<br>• 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 template (Google Slides)<br>• Alice's Restaurant starter file (WeScheme)<br>• Restaurant - Intro (Original (Page 48), Solution (Page 48))<br>• Restaurant - Explore (Original (Page 49), Solution (Page 49)) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Domain & Range of Piecewise Functions (Desmos Activity) |
| **Key Points for the Facilitator** | • The Design Recipe looks a bit different for piecewise, or ***conditional***, ***function***s. Check that students are taking time to write ***example***s and circle what is changing. |

For a textbook-like version of materials similar to these, you may wish 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!")

---

# Warmup

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.

# 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*

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, and turn to Welcome to Alice's Restaurant! (Page 48).

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 49) 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]))
```

---

# Additional Exercises:

- 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.

# Welcome to Alice's Restaurant!

Alice has hired you to improve some code used at the restaurant. The code we'll be improving on is shown below.

**Read through the code line-by-line with your partner before writing down your observations in the tables below.**

```
; cost : String -> Number
; given a item, produce the cost of that item
(define (cost item)
  (cond
    [(string=? item "hamburger")     6.00]
    [(string=? item "onion rings")   3.50]
    [(string=? item "fried tofu")    5.25]
    [(string=? item "pie")           2.25]
    [else "Sorry, that's not on the menu!"]))
```

| | |
|---|---|
| 1) **I notice...** | 2) **I wonder...** |
| 3) **Familiar things I see in the code** | 4) **Unfamiliar things I see in the code** |

# Alice's Restaurant - Explore

Alice's code has some new elements we haven't seen before, so let's experiment a bit to figure out how it works! **Open the "Alice's Restaurant starter file, click "Run", and try using the `cost` function in the Interactions window.**

1) What does `(cost "hamburger")` evaluate to? _____

2) What does `(cost "pie")` evaluate to? _____

3) What if you ask for `(cost "fries")` ? _____

4) Explain what the function is doing in your own words.

_____

_____

_____

5) What is the function's name? _____ Domain? _____ Range? _____

6) What is the name of its variable? _____

7) Alice says onion rings have gone up to $3.75. Change the `cost` function to reflect this.

8) Try adding menu items of your own. What's your favorite?

9) For an unknown food item, the function produces the String `"That's not on the menu!"`

Is this a problem? Why or why not?

_____

_____

10) Suppose Alice wants to calculate the price of a hamburger, *including a 5% sales tax* . Draw a Circle of Evaluation for the expression below.

# Word Problem: order

**Directions**: Alice's Restaurant has hired you as a programmer. They offer the following menu items: hamburger ($6.00), onion rings ($3.50), fried tofu ($5.25) and pie ($2.25). Write a function called `order` which takes in the name of a menu item and outputs the price of that item.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

( _____

[ _____ _____ ]

[ _____ _____ ]

[ _____ _____ ]

[ _____ _____ ]

[ _____ _____ ]

)) _____

*what the function does with those variable(s)*

# Word Problem: update-player

**Directions**: The player moves up and down by 20 pixels each time. Write a function called `update-player`, which takes in the player's y-coordinate and the name of the key pressed ("up" or "down"), and returns the new y-coordinate.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

( _____

[ _____ _____ ]

[ _____ _____ ]

[ _____ _____ ]

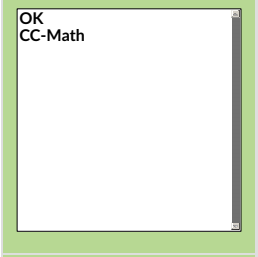
)) _____

*what the function does with those variable(s)*

# Player Animation

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | Piecewise Functions |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• 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 (Google Slides)<br>• Word Problem: update-player (Page 51) |
| **Preparation** | • Make sure all materials have been gathered<br>• Decide how students will be grouped in pairs |
| **Supplemental Resources** | • Domain & Range of Piecewise Functions (Desmos Activity) |
| **Key Points for the Facilitator** | • Encourage students to challenge themselves when creating update-player by completing one of the extension activities.<br>• 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!<br>• Adding comments to code - if you have to ask a student "What are you trying to do there?", then they probably need more comments! |

For a textbook-like version of materials similar to these, you may wish 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

---

# Warmup

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.

---

# 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

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`?**

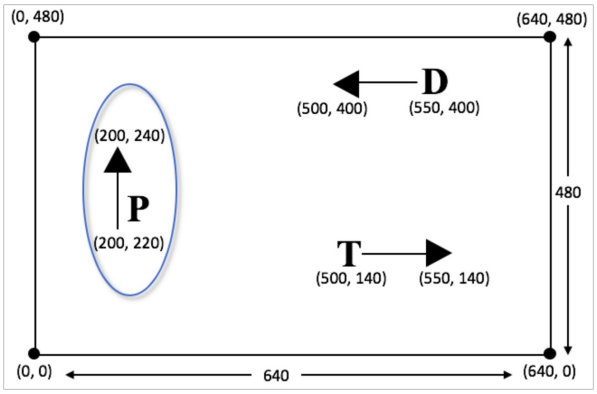
  *The Name is `update-player`, the Domain consists of a Number and String, and Range is a Number.*

- **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`*

- **What should happen mathematically to the y-coordinate of `PLAYER` when the user presses the "up" key?**

  *It should increase, the program should add something to it*




Students complete Word Problem: update-player (Page 51) with a partner, then type their code into their **Game Project file** and test.

## Possible 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?
- What happens if your little brother or sister walks by and hits a random key that doesn't have a meaning in your function? What *should* happen?

---

# 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 52) 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?

### 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.

# Challenges for update-player

For each of the challenges below, see if you can come up with two EXAMPLEs of how it should work!

1) **Warping** - Program one key to "warp" the player to a set location, such as the center of the screen.

```
(EXAMPLE (update-player _____ ) _____ )

(EXAMPLE (update-player _____ ) _____ )
```

2) **Boundaries** - Change `update-player` such that `PLAYER` cannot move off the top or bottom of the screen.

```
(EXAMPLE (update-player _____ ) _____ )

(EXAMPLE (update-player _____ ) _____ )
```

3) **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.

```
(EXAMPLE (update-player _____ ) _____ )

(EXAMPLE (update-player _____ ) _____ )
```

4) **Hiding** - Add a key that will make `PLAYER` seem to disappear, and reappear when the same key is pressed again.

```
(EXAMPLE (update-player _____ ) _____ )

(EXAMPLE (update-player _____ ) _____ )
```

# Word Problem: line-length

**Directions**: Write a function called 'line-length', which takes in two numbers and returns the **positive difference** between them. It should always subtract the smaller number from the bigger one. If they are equal, it should return zero.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE (line-length 10 5) (- 10 5))

*function name* *input(s)* *what the function produces*

(EXAMPLE (line-length 2 8) (- 8 2))

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

```
(
  [                                   ]
  [                                   ]
))
```

*what the function does with those variable(s)*

# 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.

| | |
|---|---|
| **Prerequisites** | Piecewise Functions |
| **Relevant Standards** <select size="4"><option>OK</option><option>CC-Math</option></select> | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Explain how the distance formula is related to the Pythagorean theorem.<br>• Write a function for the distance formula. |
| **Student-Facing Lesson Goals** | • I can explain how the distance formula is connected to the Pythagorean theorem.<br>• I can write a function that takes in 2 points and returns the distance between them. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Multiple Representations (Page 55) (PDF)<br>• Design Recipe: Distance - Original (Page 56), Solution (Page 56) |
| **Supplemental Resources** | • Absolute Value (Desmos)<br>• Absolute Value Inequality Illustrator (Geogebra)<br>• Absolute Value (Quizizz)<br>• Distance Formula (Geogebra)<br>• Distance Formula (Quizizz)<br>• Pythagorean Theorem (Quizizz)<br>• Pythagorean Theorem (Geogebra) |
| **Preparation** | • Make sure all materials have been gathered<br>• 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. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

**circle of evaluation ::** a diagram of the structure of an expression (arithmetic or code)

**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

---

# Warmup

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

# 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*

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.*

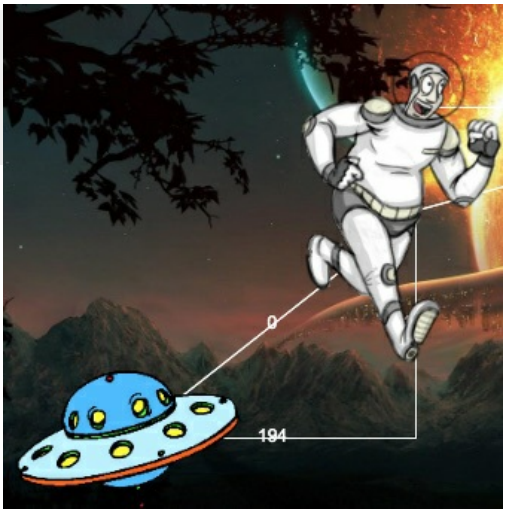
- What does it mean for characters to 'hit' one another? To collide?

  *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 hypoteneuse 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 explore using the `line-length` function in the ***Interactions area***.

### 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?

---

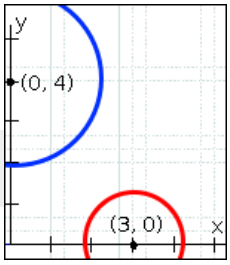
# Distance in 2 Dimensions

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!

### 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 hypoteneuse 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 The Distance Between (0, 2) and (4, 5) (Page 54) in your student workbook. Convert this expression to a Circle of Evaluation, and then to code.

Optional: Have students use this Graphic Organizer (Page 55) to model the distance formula for these coordinates with the Circles of Evaluation.

empty

> Using Word Problem: distance (Page 56), write a function that takes in two ***coordinate*** pairs (four numbers) of two characters (x1, y1) and (x2, y2) and returns the distance between those two points. _HINT: the code you wrote in The Distance Between (0, 2) and (4, 5) (Page 54) 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.

### *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 student to look back at what they circled and labeled in the examples step. *This is why we label!*

### *Synthesize*

---

# Additional Exercises:

- Bootstrap: Algebra - More Design Recipe Practice (Desmos Activity)

# The Distance Between (0, 2) and (4, 5)

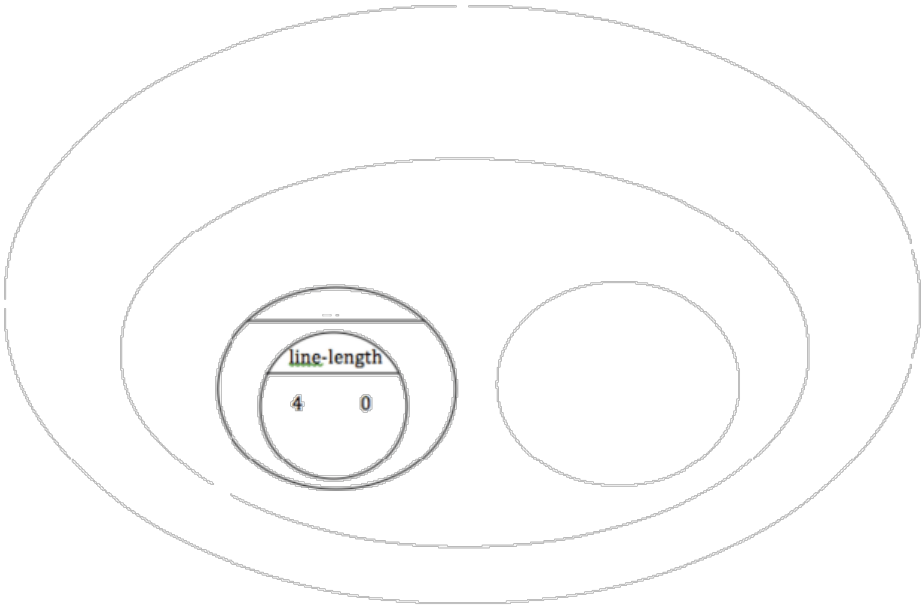
The distance between $x_1$ and $x_2$ is computed by `(line-length x1 x2)`. The distance between $y_1$ and $y_2$ is computed by `(line-length y1 y2)`. Below is the equation to compute the hypotenuse of a right triangle with those amount for legs:

$$\sqrt{line\text{-}length(x_1, x_2)^2 + line\text{-}length(y_1, y_2)^2}$$

Suppose your player is at **(0, 2)** and a character is at **(4, 5)**. What is the distance between them? With your pencil, label which numbers represent $x_1$, $y_1$, $x_2$ and $x_y$. The equation to compute the distance between these points is:

$$\sqrt{line\text{-}length(0, 4)^2 + line\text{-}length(2, 5)^2}$$

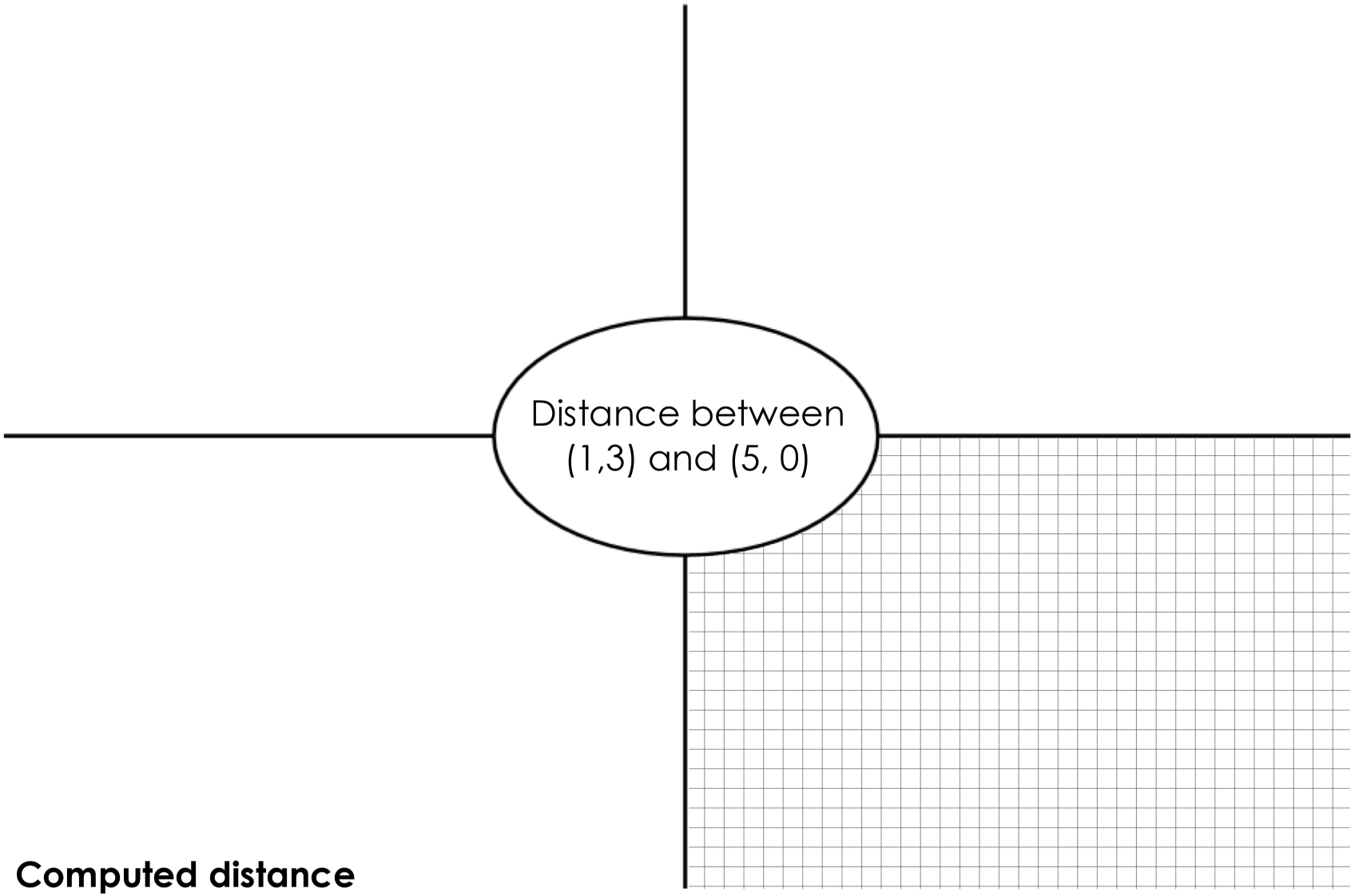
**1. Translate the expression above, for (0,2) and (4,5) into a Circle of Evaluation below.**




**2. Convert the Circle of Evaluation to Code below.**

**Circle of Evaluation**

**Code**




**Computed distance between (1, 3) and (5, 0)**

**Graph**

# Word Problem: distance

**Directions**: Use the Design Recipe to write a function `distance`, which takes in FOUR inputs: `px` and `py` (the x- and y-coordinate of the Player) and `cx` and `cy` (the x- and y-coordinates of another character). coordinates of two objects and produces the distance between them in pixels.

## Contract and Purpose Statement

*Every contract has three parts…*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes…*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values…*

(define ( _____ _____ )

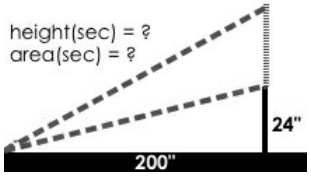
*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Top Down / Bottom Up

A retractable flag pole starts out 24 inches tall, and grows taller at a rate of 0.6in/sec. An elastic is anchored 200 inches from the base and attached to the top of the pole, forming a right triangle. Using a top-down or bottom-up strategy, define functions that compute the *height* of the pole and the *area* of the triangle after a given number of seconds.




**Directions**: Define your first function ( `height` or `area` ) here.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

---

**Directions**: Define your second function ( `height` or `area` ) here.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

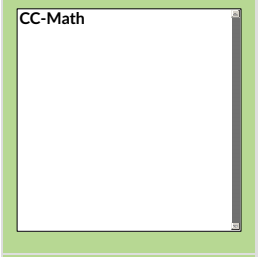
_____ )

*what the function does with those variable(s)*

# Collision Detection

(Also available for Pyret)

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

| | |
|---|---|
| **Prerequisites** | Piecewise Functions |
| **Relevant Standards**   | *Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).* |
| **Lesson Goals** | Students will be able to:<br>• Explain how the distance formula is related to the Pythagorean theorem.<br>• Write a function for the distance formula. |
| **Student-Facing Lesson Goals** | • I can explain how the distance formula is connected to the Pythagorean theorem.<br>• I can write a function that takes in 2 points and returns the distance between them. |
| **Materials** | • Lesson slides template (Google Slides)<br>• Sample game file - no distance lines (WeScheme)<br>• Sample game file - with distance lines (WeScheme)<br>• original (Page 58) |
| **Supplemental Resources** | • Absolute Value (Desmos)<br>• Absolute Value Inequality Illustrator (Geogebra)<br>• Absolute Value (Quizizz)<br>• Distance Formula (Geogebra)<br>• Distance Formula (Quizizz)<br>• Pythagorean Theorem (Quizizz)<br>• Pythagorean Theorem (Geogebra) |
| **Preparation** | • Make sure all materials have been gathered<br>• 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. |

For a textbook-like version of materials similar to these, you may wish to see the prior unit-based version

***Glossary***

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

**circle of evaluation ::** a diagram of the structure of an expression (arithmetic or code)

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

---

# Warmup

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

# Problem Decomposition Returns! 20 minutes

## *Overview*

Students revisit the problem decomposition concept from earlier lessons.

## *Launch*

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 57) 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 58), have students write a function that takes in two coordinate pairs (four numbers) of two characters (x1, y1) and (x2, y2) and returns a ***Boolean*** as to whether or not the two characters have gotten within 50 ***pixels*** of each other.

### *Synthesize*

- Since students started out with the `distance` function first, which strategy are they using to decompose collision detection?
- 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 characters that are much taller than they are wide (or wider than they are tall!), using the radius to determine collision won't work very well. Have students compute the Manhattan Distance to take the more-rectangular dimensions of their characters.

# Word Problem: collide?

**Directions**: Use the Design Recipe to write a function `collide?`, which takes in the coordinates of two objects and checks if they are close enough to collide.

## Contract and Purpose Statement

*Every contract has three parts...*

; _____ : _____ -> _____

*function name* *domain* *range*

; _____

*what does the function do?*

## Examples

*Write some examples, then circle and label what changes...*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

(EXAMPLE ( _____ _____ ) _____ )

*function name* *input(s)* *what the function produces*

## Definition

*Write the definition, giving variable names to all your input values...*

(define ( _____ _____ )

*function name* *variable(s)*

_____ )

*what the function does with those variable(s)*

# Contracts

Contracts tell us how to use a function. For example: `ellipse : Number, Number, String, String -> Image` tells us that the name of the function is `ellipse`, and that it takes four inputs (two Numbers and two Strings). From the contract, we know `(ellipse 100 50 "outline" "red")` will evaluate to an Image.

| Name | | Domain | | Range |
|---|---|---|---|---|
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |

# Contracts

Contracts tell us how to use a function. For example: `ellipse : Number, Number, String, String -> Image` tells us that the name of the function is `ellipse`, and that it takes four inputs (two Numbers and two Strings). From the contract, we know `(ellipse 100 50 "outline" "fuchsia")` will evaluate t an Image.

| Name | | Domain | | Range |
|---|---|---|---|---|
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |

# Contracts

Contracts tell us how to use a function. For example: `ellipse : Number, Number, String, String -> Image` tells us that the name of the function is `ellipse` , and that it takes four inputs (two Numbers and two Strings). From the contract, we know `(ellipse 100 50 "outline" "teal")` will evaluate to ar Image.

| Name | | Domain | | Range |
|---|---|---|---|---|
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |

# Contracts

Contracts tell us how to use a function. For example: `ellipse : Number, Number, String, String -> Image` tells us that the name of the function is `ellipse` , and that it takes four inputs (two Numbers and two Strings). From the contract, we know `(ellipse 100 50 "solid" "darkgreen")` will evaluate t an Image.

| Name | | Domain | | Range |
|---|---|---|---|---|
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |
| ; | : | | -> | |
| ; | | | | |