Name: _____

Teacher Materials

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

Bootstrap is licensed under a Creative Commons 3.0 Unported License. Based on a work from www.BootstrapWorld.org. Permissions beyond the scope of this license may be available at contact@BootstrapWorld.org.

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!

Notice and Wonder

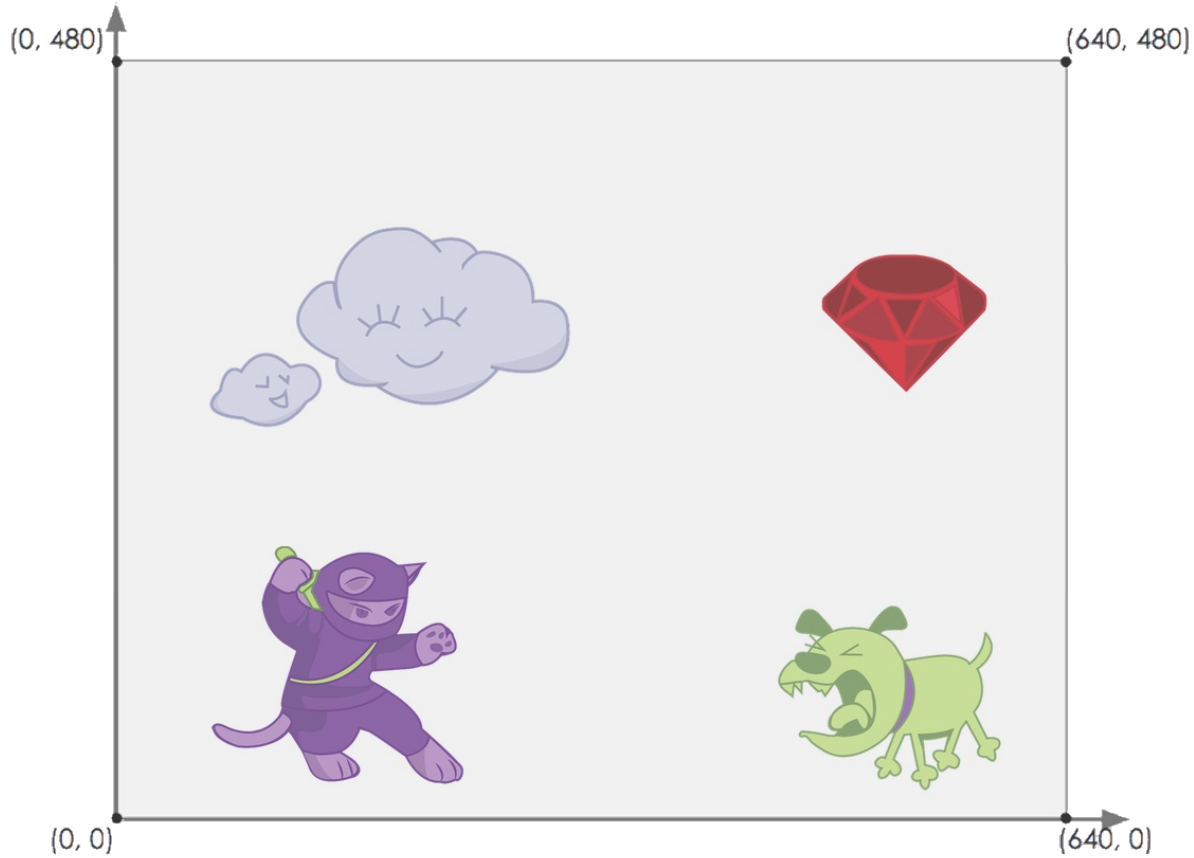
Write down what you notice and wonder about the Ninja Cat game screenshot.

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

What do you Notice?	What do you Wonder?

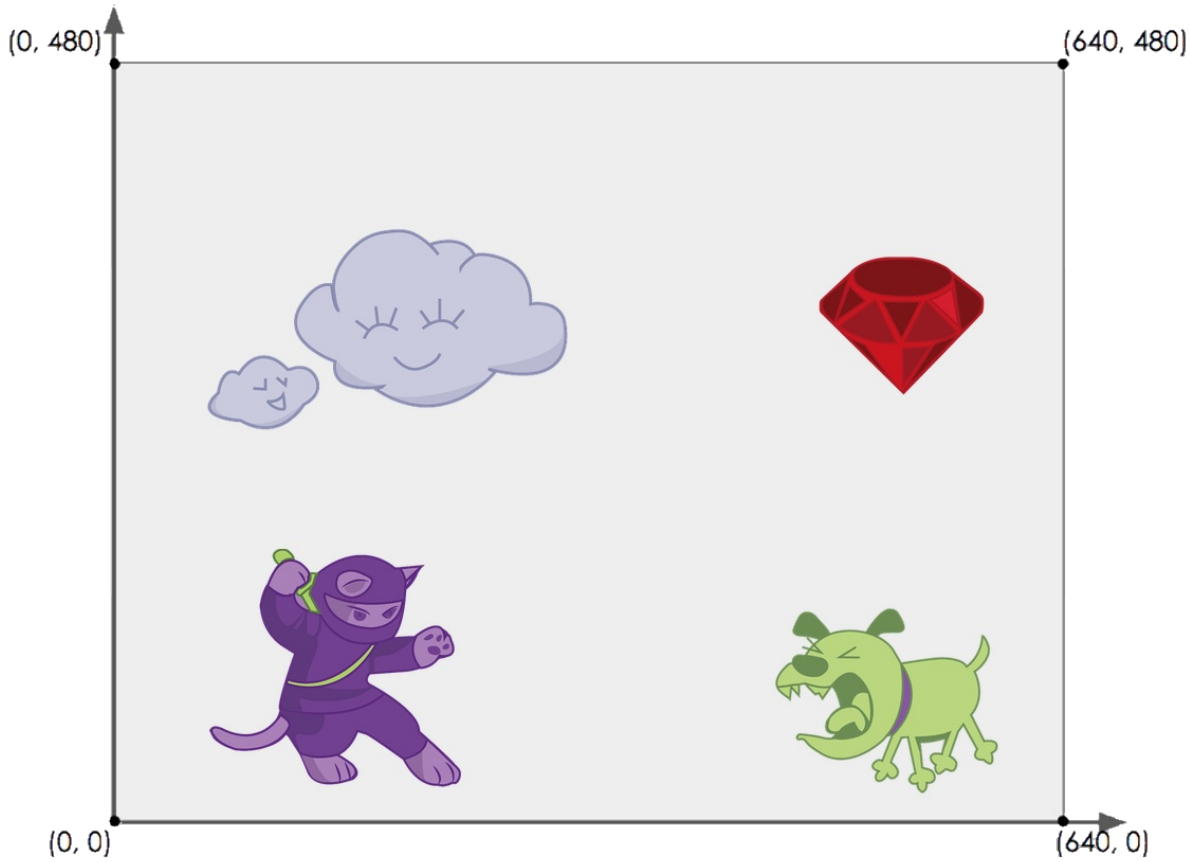
Reverse Engineer a Video Game

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
Cloud	*Position*	*x-coordinate*
Ruby	*Position*	*x-coordinate*
NinjaCat	*Position*	*x-coordinate & y-coordinate*
Score	*Value*	*Number*

Estimating Coordinates



Answers will vary. Most important is that students use the same x-coordinate for the Dog and the Ruby.

The coordinates for the PLAYER (NinjaCat) are: (*160* , *80*)
x y

The coordinates for the DANGER (Dog) are: (*530* , *70*)
x y

The coordinates for the TARGET (Ruby) are: (*530* , *310*)
x y

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.

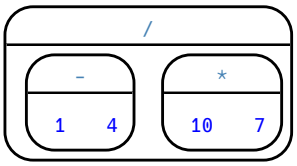
The Danger is a _____
The Danger moves only to the left or right.

Artwork/Sketches/Proof of Concept

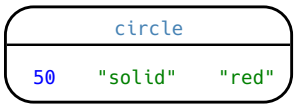
Draw a rectangle representing your game screen, and label the bottom-left corner as the coordinate (0,0). Then label the other four corners. Then, in the rectangle, sketch a picture of your game!

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 write whatever is on the left of the circle, then the **function** at the top, and then whatever is on the 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!

Notice and Wonder

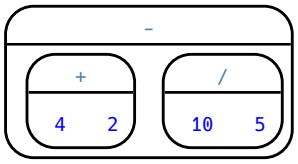
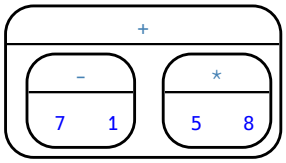
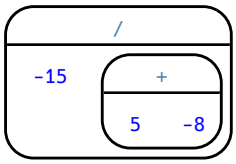
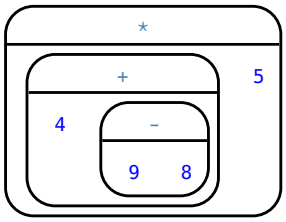
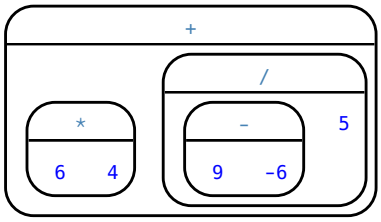
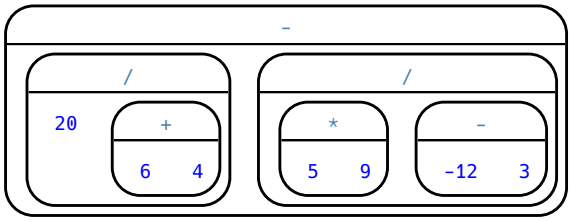
Try typing numbers into the Interactions Area, hitting "Enter", and see what you get back! Some ideas:

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?
(sample response) I notice that... *- fractions are turned into decimals* *- if you click on a decimal, you get the fraction back* *- decimals are written with a zero before the decimal point* *- repeating decimals are barred rather than rounded* *- I got tired of making my numbers bigger before the computer did* *- negative numbers work*	

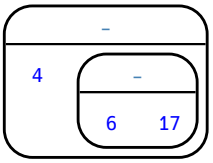
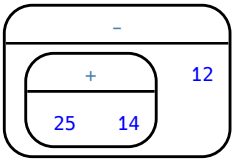
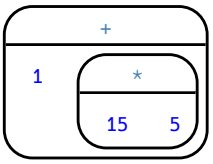
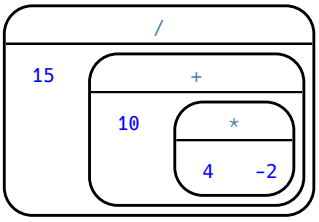
Completing Circles of Evaluation from Arithmetic Expressions

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	$\frac{20}{6 + 4} - \frac{5 \times 9}{-12 - 3}$	

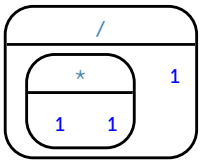
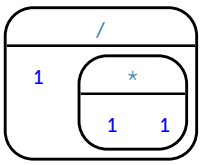
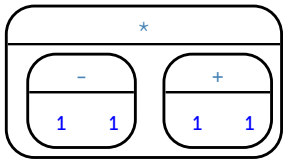
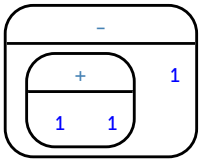
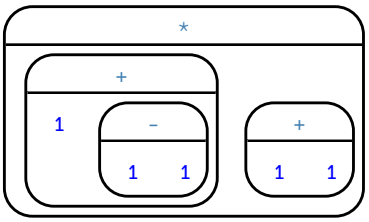
Creating Circles of Evaluation from Arithmetic Expressions

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	$\dfrac{15}{10 + 4 \times -2}$	

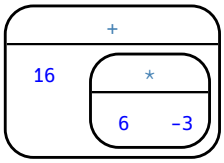
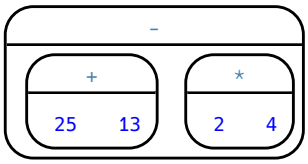
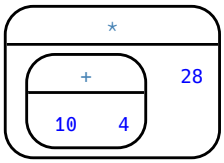
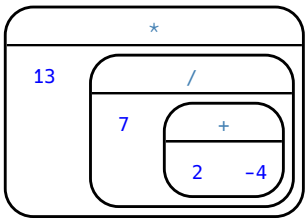
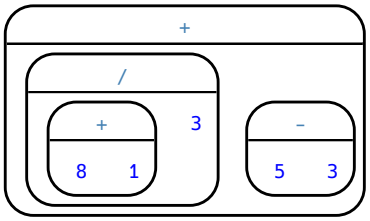
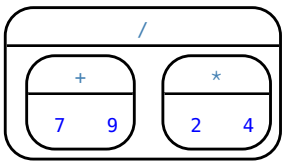
Matching Circles of Evaluation and Arithmetic Expressions

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** -C	**A**	$\frac{1}{1 \times 1}$
	**2** -A	**B**	$1 + 1 - 1$
	**3** -E	**C**	$\frac{1 \times 1}{1}$
	**4** -B	**D**	$(1 + (1 - 1)) \times (1 + 1)$
	**5** -D	**E**	$(1 - 1) \times (1 + 1)$

Completing Partial Code from Circles of Evaluation

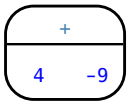
For each Circle of Evaluation on the left, finish the Code on the right by filling in the blanks.

	Circle of Evaluation	Code
1		`16 + (6 * -3)`
2		`(25 + 13) - (2 * 4)`
3		`(10 + 4) * 28`
4		`13 * (7 / (2 + -4))`
5		`((8 + 1) / 3) + (5 - 3)`
6		`(7 + 9) / (2 * 4)`

Translating Circles of Evaluation to Code

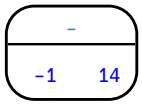
Translate the Circles of Evaluation into Code.

1)



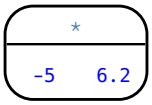
4 + -9

2)



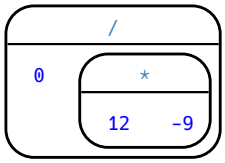
-1 - 14

3)



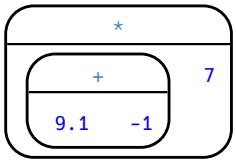
-5 * 6.2

4)



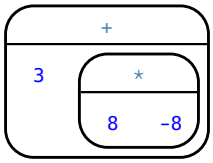
0 / (12 * -9)

5)



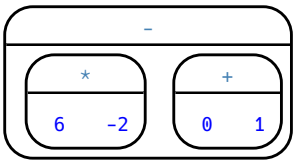
(9.1 + -1) * 7

6)



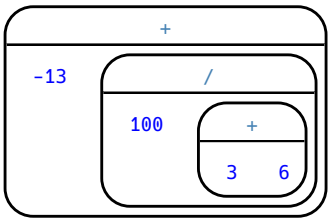
3 + (8 * -8)

7)



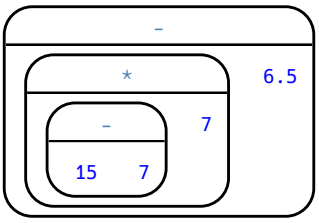
(6 * -2) - (0 + 1)

8)



-13 + (100 / (3 + 6))

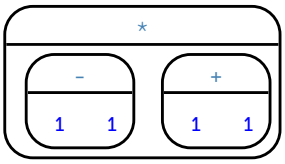
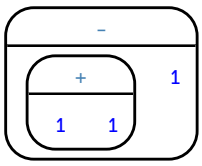
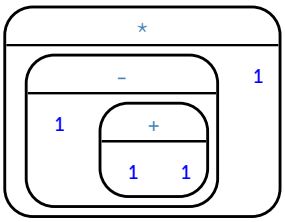
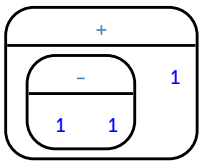
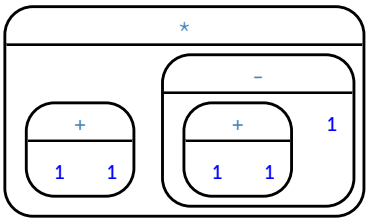
9)



((15 - 7) * 7) - 6.5

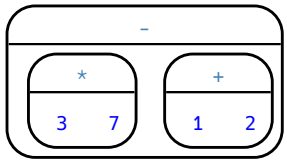
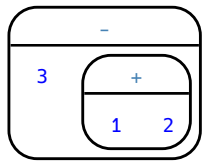
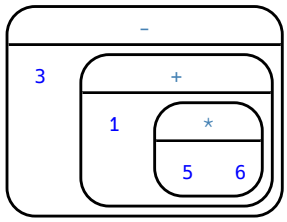
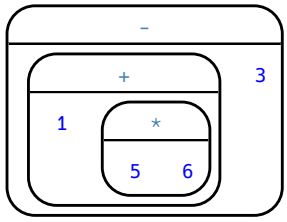
Matching Circles of Evaluation & Code

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

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

Arithmetic Expressions to Circles of Evaluation & Code

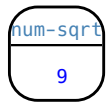
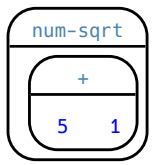
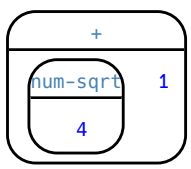
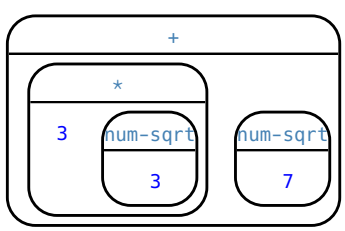
Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code.

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

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 `num-sqrt` .

	Arithmetic	Circle of Evaluation	Code
1			`num-sqrt(9)`
2			`num-sqrt(5 + 1)`
3			`num-sqrt(4) + 1`
4			`(3 * num-sqrt(3)) + num-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` `50` `"solid"` `"black"`	produces →	
	produces →	
	produces →	
	produces →	
	produces →	

Mystery Functions!

2) There is a function called `regular-polygon` with 4 inputs. What do they mean?

The first input is the size. The second input is the number of sides of the regular polygon.

The third input is whether the figure is solid or outline. The fourth input is the color.

3) There is a function called `radial-star` with 5 inputs. What do they mean?

The first input is the number of points in the star.

The second and third inputs are the outer and inner radius of the star.

The fourth input is whether the star is solid or outline. The fifth input is the color.

4) There is a function called `text`. Try to figure out how to use it! What do the inputs mean?

The first input (a String) is the string we want to display.

The second input (a Number) tells us how big to print that string.

The third input tells us what the color will be.

Reading for Domain and Range

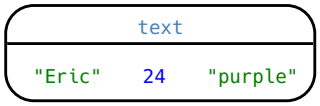
1) What is the **name** of the function being used in: `string-length("broccoli") + 8`	`string-length`
2) What is the **domain** of the outermost function being used in: `scale(2, circle(40, "solid", "blue"))`	*Number*, *Image*
3) What is the **domain** of the innermost function being used in: `scale(2,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

Composing Image Functions

You'll be investigating these functions with your partner:

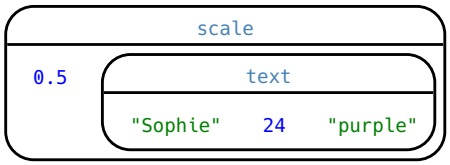
```
# text :: String, Number, String -> Image
# scale :: Number, Image -> Image
# rotate :: Number, Image -> Image
# flip-horizontal :: Image -> Image
# flip-vertical :: Image -> Image
```

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.




```
text("Eric", 24, "purple")
```

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.




```
scale(0.5, text("Sophie", 24, "purple"))
```

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

When the number is less than 1, it scales the image down to a smaller version

When the number is greater than 1, it scales the image up to a larger version

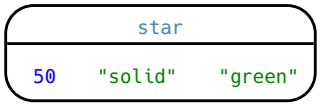
4) Try out `rotate`, `flip-horizontal`, and `flip-vertical`. Use the space below to write your code, then test out your code in WeScheme when you're ready.

For sample code for these functions, refer to the contracts page of the teachers manual.

Function Composition—Practice

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

Circle of Evaluation:

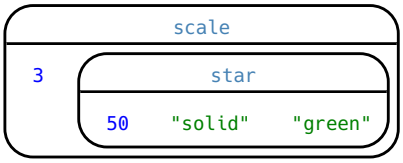


Code: `star(50, "solid", "green")`

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`)

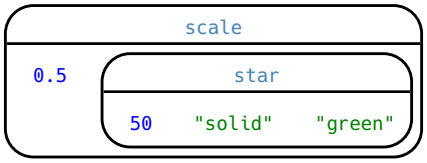
Circle of Evaluation:



Code: `scale(3, star(50, "solid", "green"))`

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

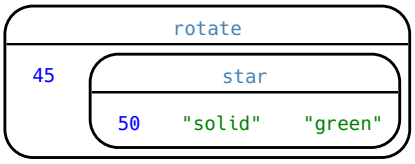
Circle of Evaluation:



Code: `scale(0.5, star(50, "solid", "green"))`

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

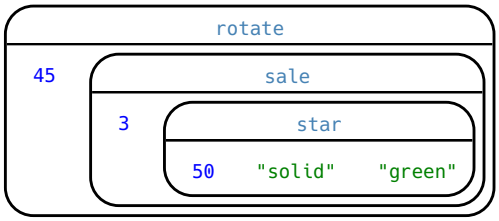
Circle of Evaluation:



Code: `rotate(45, star(50, "solid", "green"))`

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

Circle of Evaluation:



Code: `rotate(45, scale(3, star(50, "solid", "green")))`

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 `var x = 5 + 1` . Here are a few value definitions:

```
x = 5 + 1
y = x * 7
food = "Pizza!"
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 — Explore!

```
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?

A solid, red triangle appears

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

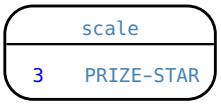
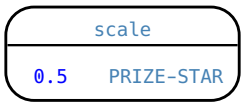
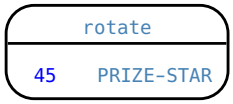
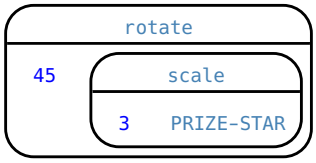
Ideas: `eye-color` (a String), `age` (a Number), `fav-shape` (an Image)

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.

```
var PRIZE-STAR = star(65, "outline", "pink")
```

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`)  `scale ( 3, PRIZE-STAR )`	3) The outline of a pink star that is half the size of the original (using scale)  `scale ( 0.5, PRIZE-STAR )`
4) The outline of a pink star that is size 65 and has been rotated 45 degrees  `rotate ( 45, PRIZE-STAR )`	5) The outline of a pink star that is 3 times the size of the original **and** has been rotated 45 degrees  `rotate ( 45, scale ( 3, PRIZE-STAR ) )`

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

(Sample response) Defining a value that you will be using repeatedly allows you to use the value as shorthand for the part of the code you use again and again.

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?

Mapping Examples with Circles of Evaluation

Contract: _____

Purpose Statement: _____

If I type...	→	It should map to...

EXAMPLE #1: Circle of Evaluation
`gt`
`75`
 	
Code: `gt(75)`
	→	Circle of Evaluation:
		`triangle`
		`75` `"solid"` `"green"`
		
		Code: `triangle(75, "solid", "green")`

EXAMPLE #2: Circle of Evaluation Code:	→	Circle of Evaluation: Code:

Fast Functions

```
# gt:: Number -> Image

examples:
  gt ( 10 ) is triangle(10, "solid", "green")

  gt ( 20 ) is triangle(20, "solid", "green")

end

fun gt( size ):

  triangle(size, "solid", "green")

end
```

```
# gold-star:: Number -> Image

examples:
  gold-star ( 35 ) is star(35, "solid", "gold")

  gold-star ( 27 ) is star(27, "solid", "gold")

end

fun gold-star( radius ):

  star(radius, "solid", "gold")

end
```

```
# :: -> 

examples:
  ( ) is 

  ( ) is 

end

fun ( ):

end
```

```
# :: -> 

examples:
  ( ) is 

  ( ) is 

end

fun ( ):

end
```

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...

`rocket-height`:: `Number` -> `Number`

function name *domain* *range*

Takes in seconds since liftoff and returns the height of the rocket, which is traveling 7 m/s

what does the function do?

Examples

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

examples:

`rocket-height` (`0`) **is** `7 * 0`

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

`rocket-height` (`3`) **is** `7 * 3`

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

end

Definition

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

fun `rocket-height`(`seconds`):

function name *variable(s)*

`7 * seconds`

what the function does with those variable(s)

end

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:

Mapping Examples with Circles of Evaluation

Contract: _____

Purpose Statement: _____

If I type...	→	It should map to...
EXAMPLE #1: Circle of Evaluation Code:	→	Circle of Evaluation: Code:
EXAMPLE #2: Circle of Evaluation Code:	→	Circle of Evaluation: 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...

```
# marquee:: String -> Image
```

function name — *domain* — *range*

```
# Takes in a message and returns an image of it in large gold letters
```

what does the function do?

Examples

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

```
examples:
  marquee ("Hooray!") is text("Hooray!", 70, "gold")
  marquee ("Marquee works") is text("Marquee works", 70, "gold")
end
```

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

Definition

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

```
fun marquee(message):
  text(message, 70, "gold")
end
```

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...

```
# circle-area:: Number -> Number
```

function name — *domain* — *range*

```
# Takes in the radius, squares it, multiplies it by pi and returns the area
```

what does the function do?

Examples

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

```
examples:
  circle-area (1) is 3.14 * num-sqr(1)
  circle-area (3) is 3.14 * num-sqr(3)
end
```

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

Definition

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

```
fun circle-area(radius):
  3.14 * num-sqr(radius)
end
```

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...

`minimum-wage`:: `Number` -> `Number`

function name *domain* *range*

Takes in a number of hours, multiplies it by $10.25 and returns that value

what does the function do?

Examples

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

examples:

`minimum-wage` (`0`) **is** `0 * 10.25`

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

`minimum-wage` (`30`) **is** `30 * 10.25`

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

end

Definition

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

fun `minimum-wage`(`hours`):

function name *variable(s)*

`hours * 10.25`

what the function does with those variable(s)

end

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...

`tip-calculator`:: `Number` -> `Number`

function name *domain* *range*

Takes in the cost of a meal, multiplies it by 0.15 and returns the value of the tip

what does the function do?

Examples

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

examples:

`tip-calculator` (`10`) **is** `0.15 * 10`

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

`tip-calculator` (`35`) **is** `0.15 * 35`

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

end

Definition

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

fun `tip-calculator`(`cost`):

function name *variable(s)*

`0.15 * cost`

what the function does with those variable(s)

end

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...

```
# globo-gym:: Number -> Number
```

function name — *domain* — *range*

```
# Takes in a number of months and multiplies it by $45 and adds $150 and returns that value
```

what does the function do?

Examples

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

```
examples:
  globo-gym ( 0 ) is 150 + (0 * 45)
  globo-gym ( 3 ) is 150 + (3 * 45)
end
```

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

Definition

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

```
fun globo-gym( months ):
  150 + (months * 45)
end
```

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...

```
# rideshare:: Number -> Number
```

function name — *domain* — *range*

```
# Takes in a number of of miles, multiplies it be 1.50 and then adds it to 2.50
```

what does the function do?

Examples

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

```
examples:
  rideshare ( 0 ) is 2.5 + (0 * 1.5)
  rideshare ( 3 ) is 2.5 + (3 * 1.5)
end
```

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

Definition

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

```
fun rideshare( miles ):
  2.5 + (miles * 1.5)
end
```

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...

```
# moving:: Number, Number -> Number
```

function name — *domain* — *range*

```
# Takes in a number of days and multiplies it by $45, then takes in a number of miles and multiplies it by
  $0.15, then adds the two products and returns the cost of moving
```

what does the function do?

Examples

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

```
examples:
  moving(1, 600) is (1 * 55) + (600 * 0.15)
  moving(3, 1500) is (3 * 55) + (1500 * 0.15)
end
```

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

Definition

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

```
fun moving(days, miles):
  (days * 55) + (miles * 0.15)
end
```

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...

```
# lawn-area:: Number, Number -> Number
```

function name — *domain* — *range*

```
# Takes in 2 numbers, length and width, and multiplies them and returns that value
```

what does the function do?

Examples

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

```
examples:
  lawn-area(10, 20) is 10 * 20
  lawn-area(100, 300) is 100 * 300
end
```

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

Definition

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

```
fun lawn-area(length, width):
  length * width
end
```

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...

`rect-perimeter`:: `Number, Number` -> `Number`

function name *domain* *range*

Takes in 2 numbers, length and width, and returns the double of the sum of both numbers

what does the function do?

Examples

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

```
examples:
  rect-perimeter(10, 20) is 2 * (10 + 20)
  rect-perimeter(200, 350) is 2 * (200 + 350)
end
```

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

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

Definition

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

```
fun rect-perimeter(length, width):
  2 * (length + width)
end
```

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...

`rectprism-vol`:: `Number, Number, Number` -> `Number`

function name *domain* *range*

Takes in 3 numbers, length, width, and height, and multiplies them to return that value

what does the function do?

Examples

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

```
examples:
  rectprism-vol(10, 20, 30) is 10 * (20 * 30)
  rectprism-vol(100, 250, 350) is 100 * (250 * 350)
end
```

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

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

Definition

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

```
fun rectprism-vol(length, width, height):
  length * (width * height)
end
```

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...

`split-tab`:: `Number, Number` -> `Number`

function name *domain* *range*

Takes in a cost and a number of people and divides the cost by the number of people, returning the value.

what does the function do?

Examples

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

examples:

`split-tab (200, 10) is 200 / 10`

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

`split-tab (500, 5) is 500 / 5`

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

end

Definition

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

fun `split-tab( cost, people ):`

function name *variable(s)*

`cost / people`

what the function does with those variable(s)

end

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...

`num-cube`:: `Number` -> `Number`

function name *domain* *range*

Takes in a number, cubes it and returns that value.

what does the function do?

Examples

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

examples:

`num-cube (1) is 1 * (1 * 1)`

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

`num-cube (3) is 3 * (3 * 3)`

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

end

Definition

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

fun `num-cube( n ):`

function name *variable(s)*

`n * (n * n)`

what the function does with those variable(s)

end

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...

```
# update-danger:: Number -> Number
```

function name — *domain* — *range*

```
# Consumes an x-coordinate and returns a new x-coordinate
```

what does the function do?

Examples

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

```
examples:
  update-danger ( 160 ) is 160 - 50
  update-danger ( -85 ) is -85 - 50
end
```

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

Definition

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

```
fun update-danger( x ):
  x - 50
end
```

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...

```
# update-target:: Number -> Number
```

function name — *domain* — *range*

```
# Consumes an x-coordinate and returns a new x-coordinate
```

what does the function do?

Examples

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

```
examples:
  update-target ( 130 ) is 130 + 50
  update-target ( -25 ) is -25 + 50
end
```

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

Definition

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

```
fun update-target( x ):
  x + 50
end
```

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

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!

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...

```
# revenue:: Number -> Number
```

function name — *domain* — *range*

Consumes a Number of glasses sold and produces the revenue

what does the function do?

Examples

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

```
examples:
  revenue(1) is 1.75 * 1
  revenue(5) is 1.75 * 5
end
```

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

Definition

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

```
fun revenue(glasses):
  1.75 * glasses
end
```

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...

```
# cost:: Number -> Number
```

function name — *domain* — *range*

```
# Consumes a Number of glasses sold and produces the cost
```

what does the function do?

Examples

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

```
examples:
  cost(1) is 0.3 * 1
  cost(5) is 0.3 * 5
end
```

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

Definition

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

```
fun cost(glasses):
  0.3 * glasses
end
```

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...

```
# profit:: Number -> Number
```

function name — *domain* — *range*

```
# Consumes a Number of glasses sold and produces the profit
```

what does the function do?

Examples

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

```
examples:
  profit(1) is revenue(1) - cost(1)
  profit(5) is revenue(5) - cost(5)
end
```

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

Definition

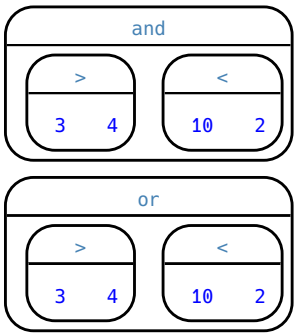
Write the definition, giving variable names to all your input values...

```
fun profit(glasses):
  revenue(glasses) - cost(glasses)
end
```

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.
  - `(3 > 4) and (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`.
  - `(3 > 4) or (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:



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 5	2) `0 > 5` will be true
3) `4 / 2` will be 2	4) `1 = 9` will be false
5) `0 - 9` will be -9	6) `2 <= 2` will be true
7) `string-length("bat")` will be 3	8) `string-equal("dog", "cat")` will be false

9) What does the operator `<` do?

it takes in two inputs, and returns true if the first one is less than the second

10) What does the function `string-equal` do?

it takes in two Strings, and returns true if they are exactly the same

11) Write the contract for `string-equal` in your Contracts page.

12) How many Numbers are there in the entire universe? *infinite*

13) How many Strings are there in the entire universe? *infinite*

14) How many Images are there in the entire universe? *infinite*

15) How many Booleans are there in the Universe? *two*

What are they?

true and false

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? *the x-coordinate*

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

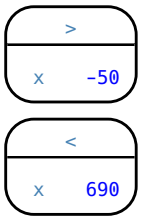
How far to the left and right can Sam go and still remain visible?

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

2) Sam is still visible on the left as long as... `x > -50`

3) Sam is still visible on the right as long as... `x < 690`

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 `is-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...

```
# is-safe-left:: Number -> Boolean
```

function name — *domain* — *range*

```
# Consumes an x-coordinate, checks to see if it is greater than -50, and produces a Boolean
```

what does the function do?

Examples

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

```
examples:
  is-safe-left ( 100 ) is 100 > -50
  is-safe-left ( -180 ) is -180 > -50
end
```

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

Definition

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

```
fun is-safe-left( x ):
  x > -50
end
```

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

Directions: Use the Design Recipe to write a function `is-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...

```
# is-safe-right:: Number -> Boolean
```

function name — *domain* — *range*

```
# Consumes an x-coordinate, checks to see if it is less than 690, and produces a Boolean
```

what does the function do?

Examples

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

```
examples:
  is-safe-right ( 250 ) is 250 < 690
  is-safe-right ( 900 ) is 900 < 690
end
```

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

Definition

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

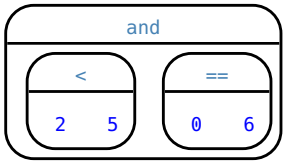
```
fun is-safe-right( x ):
  x < 690
end
```

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.

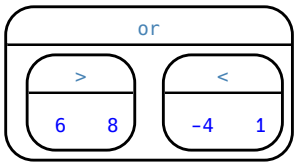
1) 2 is less than 5, and 0 is equal to 6




```
(2 < 5) and (0 == 6)
```

What will this evaluate to? false

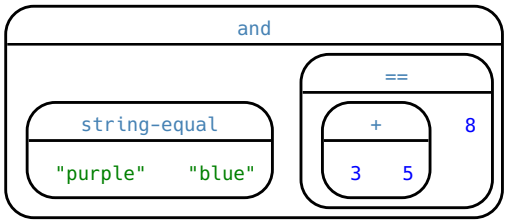
2) 6 is greater than 8, or -4 is less than 1




```
(6 > 8) or (-4 < 1)
```

What will this evaluate to? true

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




```
string-equal("purple", "blue") and ((3 + 5) == 8)
```

What will this evaluate to? false

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

```
# and :: Boolean Boolean -> Boolean
# or :: Boolean Boolean -> Boolean
```

Word Problem: is-onscreen

Directions: Use the Design Recipe to write a function `is-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...

`is-onscreen`:: `Number` -> `Boolean`

function name *domain* *range*

Consumes an x-coordinate and produces true if Sam is on the screen

what does the function do?

Examples

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

examples:

`is-onscreen` (`100`) **is** `is-safe-left(100) and is-safe-right(100)`

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

`is-onscreen` (`-180`) **is** `is-safe-left(-180) and is-safe-right(-180)`

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

end

Definition

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

fun `is-onscreen`(`x`):

function name *variable(s)*

`is-safe-left(x) and is-safe-right(x)`

what the function does with those variable(s)

end

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:

```
fun cost(pizzas):
  ask:
    | pizzas < 6 then: 10 * pizzas
    | pizzas >= 6 then: 5 * pizzas
  end
end
```

- 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.

Welcome to Alice's Restaurant!

Alice has hired you to improve some code used at the restaurant. The code we need to work 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 menu-item, produce the cost of that menu-item
fun  cost(menu-item):
  ask:
    | string-equal(menu-item, "hamburger") then:  6.00
    | string-equal(menu-item, "onion rings") then:  3.50
    | string-equal(menu-item, "fried tofu") then:  5.25
    | string-equal(menu-item, "pie") then: 2.25
    | otherwise: "Sorry, that's not on the menu!"
  end
end
```

1) **I notice...** *(sample responses) a function called* `cost`*, pipes (* `|` *symbols), a function called* `string-equal`*, numbers that look like prices, a contract and purpose statement, food items*	2) **I wonder...** *(sample responses) What is* `ask`*? Is this all that's on the menu? Can I add more food? How does the* `cost` *function work? What are the pipes (* `|` *) for? What does* `string-equal` *do?*
3) **Familiar things I see in the code** *define, contract and purpose statement, Numbers and Strings functions*	4) **Unfamiliar things I see in the code** *ask, string-equal, |-symbol, otherwise*

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? *6*

2) What does `cost("pie")` evaluate to? *2.25*

3) What if you ask for `cost("fries")` ? "Sorry, that's not on the menu!"

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

5) What is the function's name? cost Domain? String Range? Numner

6) What is the name of its variable? menu-item

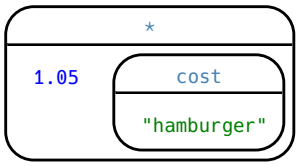
7) Alice says onion rings have gone up to $3.75. Change the `cost` function to reflect this.

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

9) For an unknown food item, the function produces a 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...

```
# order:: String -> Number
```

function name — *domain* — *range*

```
# Given a item, produce the cost of that item
```

what does the function do?

Examples

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

```
examples:
  order ("hamburger") is 6.00
  order ("pie") is 2.25
  ____ (____) is ____
  ____ (____) is ____
end
```

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

Definition

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

```
fun order(item):
  ask:
  | topping == "hamburger" then: 6.00
  | topping == "onion rings" then: 3.50
  | topping == "fried tofu" then: 5.25
  | topping == "pie" then: 2.25
  end
end
```

function name — *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...

```
# update-player:: Number, String -> Number
```

function name — *domain* — *range*

```
# Produce new y-coordinate depending on key pressed
```

what does the function do?

Examples

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

```
examples:
  update-player ( 320, "up" ) is 320 + 20
  update-player ( 100, "up" ) is 100 + 20
  update-player ( 320, "down" ) is 320 - 20
  update-player ( 100, "down" ) is 100 - 20
end
```

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

Definition

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

```
fun update-player( y, key ):
  ask:
  | key == "up"          then: y + 20
  | key == "down"        then: y - 20
  | otherwise:  y
  end
end
```

function name — *variable(s)*

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.

```
examples:
  update-player(                 ) is

  update-player(                 ) is
end
```

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

```
examples:
  update-player(                 ) is

  update-player(                 ) is
end
```

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.

```
examples:
  update-player(                 ) is

  update-player(                 ) is
end
```

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

```
examples:
  update-player(                 ) is

  update-player(                 ) is
end
```

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...

line-length:: Number, Number -> Number

function name *domain* *range*

Consumes two numbers, and produces the positive difference between them

what does the function do?

Examples

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

```
examples:
  line-length ( 10, 5 ) is 10 - 5
  line-length ( 2, 8 ) is 8 - 2
end
```

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

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

Definition

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

```
fun line-length( a, b ):
  ask:
  | a > b then: a - b
  | otherwise: b - a
  end
end
```

function name *variable(s)*

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

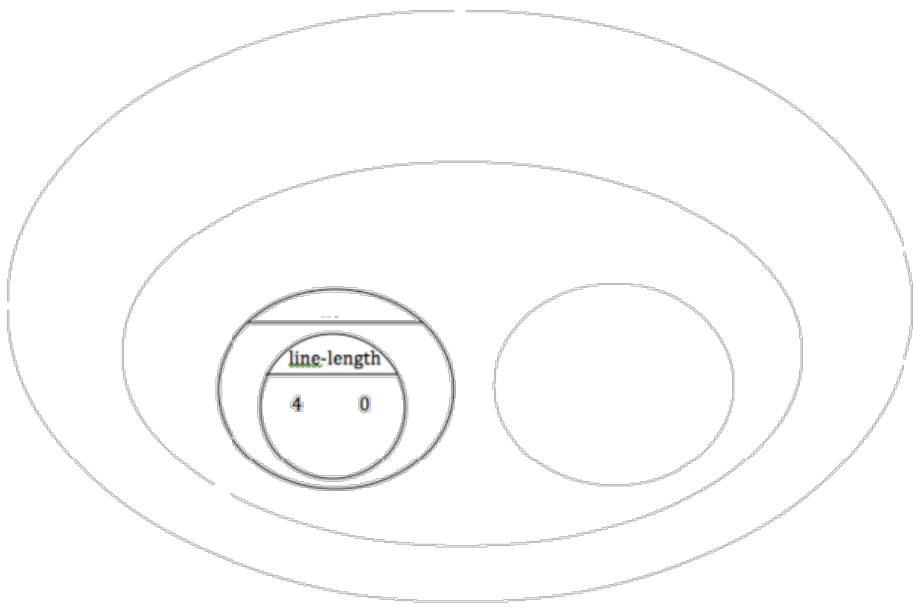
The distance between x_1 and x_2 is computed by `line-length(x1, x2)` . The distance between y_1 and y_2 is computed by `line-length(y1, y2)` . Below is the equation to compute the hypotenuse of a right triangle with those amount for legs:

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

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:

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

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

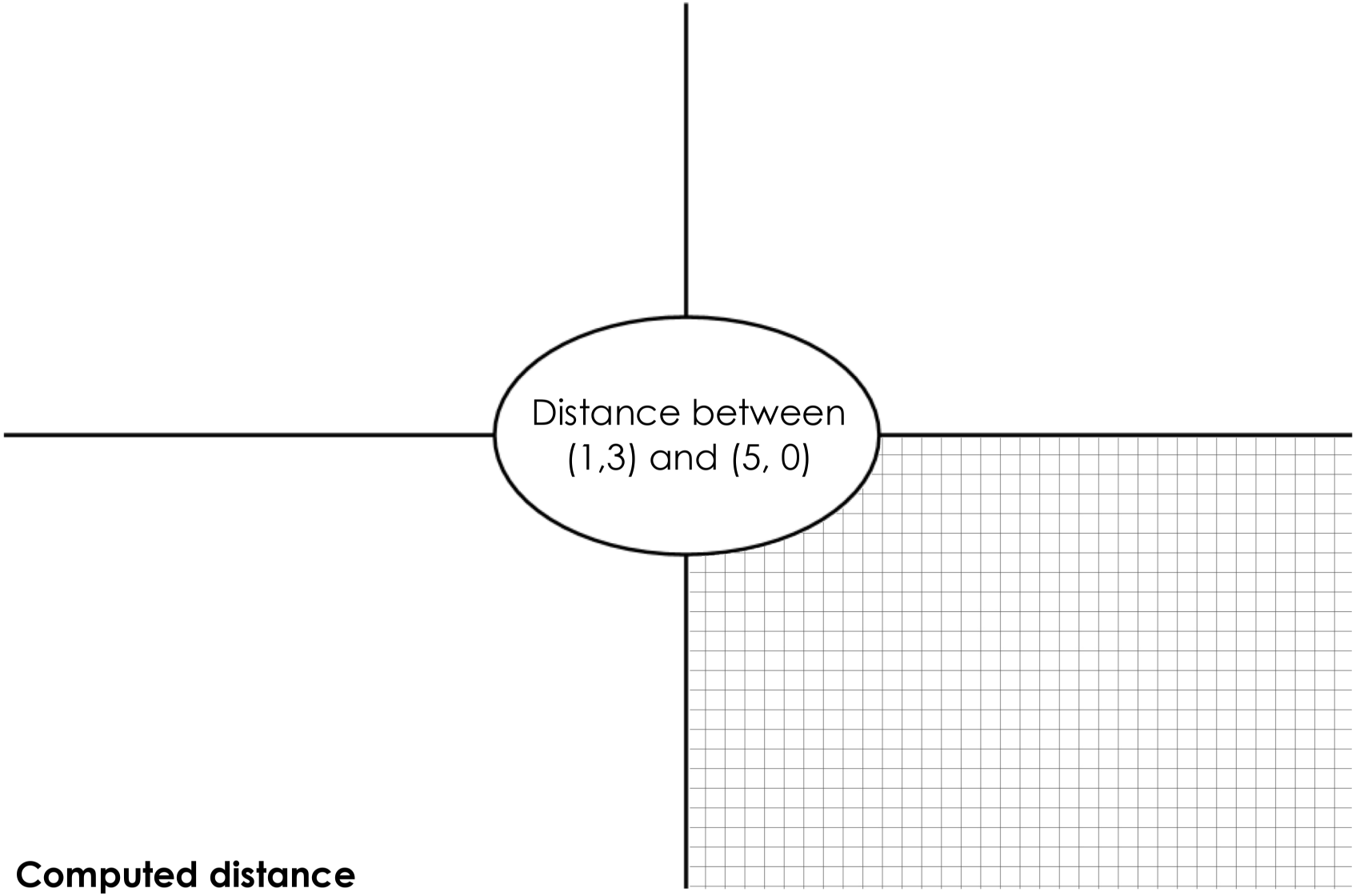


2. Convert the Circle of Evaluation to Code below .

```
num-sqrt(num-sqr(line-length(x1, x2)) + line-length(x1, x2))
```

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...

```
# distance:: Number, Number, Number, Number -> Number
```

function name — *domain* — *range*

```
# Takes in two sets of (x,y) coordinates and produces the distance between them
```

what does the function do?

Examples

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

```
examples:
  distance(0, 4, 3, 0) is num-sqrt(num-sqr(4 - 0) + num-sqr(0 - 3))
  distance(1, 30, 32, 24) is num-sqrt(num-sqr(30 - 1) + num-sqr(24 - 32))
end
```

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

Definition

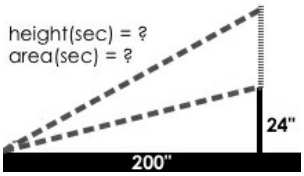
Write the definition, giving variable names to all your input values...

```
fun distance(x1, y1, x2, y2):
  num-sqrt(num-sqr(x2 - x1) + num-sqr(y2 - y1))
end
```

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...

`area`:: `Number` -> `Number`

function name *domain* *range*

Consumes seconds & produces the area of the triangle with a base of 200 and changing height

what does the function do?

Examples

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

examples:

`area` (`5`) **is** `1/2 * (200 * height(5))`

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

`area` (`6`) **is** `1/2 * (200 * height(6))`

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

end

Definition

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

fun `area(` `sec` `):`

function name *variable(s)*

`1/2 * (200 * height(sec))`

what the function does with those variable(s)

end

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

Contract and Purpose Statement

Every contract has three parts...

`height`:: `Number` -> `Number`

function name *domain* *range*

Consumes the # of seconds and produces the height, according to h=0.6s + 24

what does the function do?

Examples

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

examples:

`height` (`1`) **is** `(0.6 * 1) + 24`

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

`height` (`2`) **is** `(0.6 * 2) + 24`

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

end

Definition

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

fun `height(` `sec` `):`

function name *variable(s)*

`(0.6 * sec) + 10`

what the function does with those variable(s)

end

Word Problem: is-collide

Directions: Use the Design Recipe to write a function `is-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...

```
# is-collide:: Number, Number, Number, Number -> Boolean
```

function name — *domain* — *range*

```
# Takes in two pairs of x/y coordinates and uses the distance between them to check for collision
```

what does the function do?

Examples

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

```
examples:
  is-collide(0, 0, 3, 4) is distance(0, 0, 3, 4) < 50
  is-collide(25, 50, 250, 480) is distance(25, 50, 250, 480) < 50
end
```

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

Definition

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

```
fun is-collide(x1, y1, x2, y2):
  distance(x1, y1, x2, y2) < 50
end
```

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`, it takes four inputs (two Numbers and two Strings), and it evaluates to an `Image`. From the contract, we know `ellipse(100, 50, "outline", "red")` will evaluate to an `Image`.

Name		Domain		Range
# num-sqr	::	(Number)	->	Number
num-sqr(9)				
# num-sqrt	::	(Number)	->	Number
num-sqrt(25)				
# star	::	(Number, String, String)	->	Image
star(50, "solid", "teal")				
# circle	::	(Number, String, String)	->	Image
circle(30, "outline", "fuchsia")				
# triangle	::	(Number, String, String)	->	Image
triangle(80, "solid", "darkgreen")				
# square	::	(Number, String, String)	->	Image
square(10, "outline", "red")				
# rectangle	::	(Number, Number, String, String)	->	Image
rectangle(20, 80, "solid", "gold")				
# ellipse	::	(Number, Number, String, String)	->	Image
ellipse(30, 70, "outline", "lightblue")				
# regular-polygon	::	(Number, Number, String, String)	->	Image
regular-polygon(8, 40, "solid", "red")				
# radial-star	::	(Number, Number, String, String)	->	Image
radial-star(17, 50, 10, "solid", "orange")				

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`, it takes four inputs (two Numbers and two Strings), and it evaluates to an `Image`. From the contract, we know `ellipse(50, 100, "solid", "teal")` will evaluate to an `Image`.

Name		Domain		Range
# text	::	(String, Number, String)	->	Image
text("I'm thankful for…", 50, "green")				
# image-url	::	(String)	->	Image
image-url("https://www.bootstrapworld.org/images/icon.png")				
# scale	::	(Number, Image)	->	Image
scale(0.8, triangle(30, "solid", "red"))				
# rotate	::	(Number, Image)	->	Image
rotate(35, rectangle(30, 80, "solid", "purple"))				
# overlay	::	(Image, Image)	->	Image
overlay(star(30, "solid", "gold"),circle(30, "solid", "blue"))				
# put-image	::	(Image, Number, Number, Image)	->	Image
put-image(star(30, "solid", "red"), 50, 150, rectangle(300, 200, "outline", "black"))				
# flip-horizontal	::	(Image)	->	Image
flip-horizontal(text("Bootstrap is fun!", 60, "darkblue"))				
# flip-vertical	::	(Image)	->	Image
flip-vertical(triangle(80, "solid", "lightgreen"))				
# above	::	(Image, Image)	->	Image
above(triangle(30, "solid", "red"),square(30, "solid", "blue"))				
# beside	::	(Image, Image)	->	Image
beside(star(50, "solid", "orange"),circle(50, "solid", "green"))				

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`, it takes four inputs (two Numbers and two Strings), and it evaluates to an `Image`. From the contract, we know `ellipse(100, 50, "solid", "fuchsia")` will evaluate to an `Image`.

Name		Domain		Range
# string-equal	::	(String, String)	->	Boolean
string-equal("a tree", "trees")				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				

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`, it takes four inputs (two Numbers and two Strings), and it evaluates to an `Image`. From the contract, we know `ellipse(100, 50, "outline", "darkgreen")` will evaluate to an `Image`.

Name		Domain		Range
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				