Name: _____

Student Workbook w/Solutions

Workbook v3.0

Brought to you by the Bootstrap team:

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

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!

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* (x), *80* (y))

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

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

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 _____

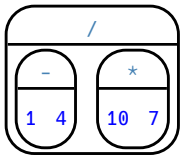
The Danger moves only to the left or right.

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!

Order of Operations

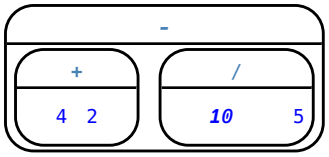
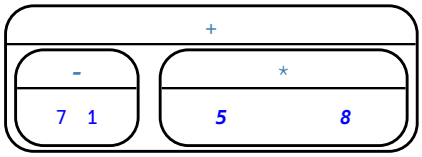
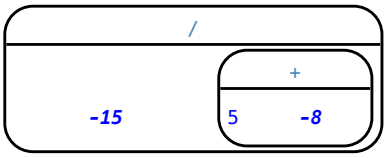
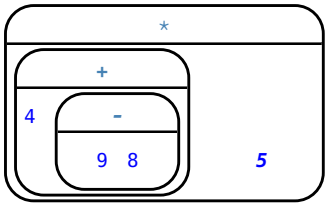
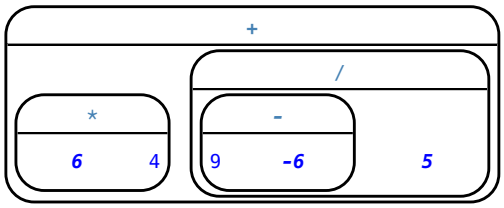
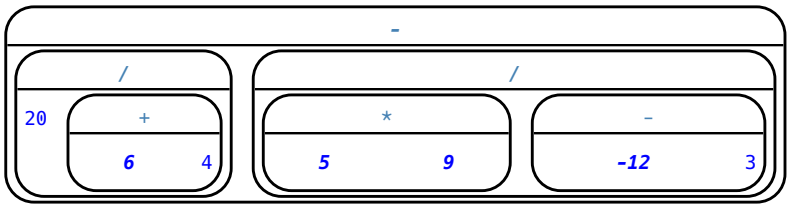
Order of Operations is incredibly important when programming. To help us organize our math into something we can trust, we can *diagram* a math expression using the **Circles of Evaluation**. For example, the expression $1 - 4 \div 10 \times 7$ can be diagrammed as shown below.



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 **operation** at the top, and then whatever is on the right. The circle above, for example, would be programmed as `((1 - 4) / (10 * 7))`.

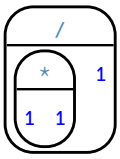
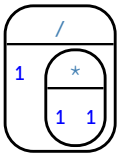
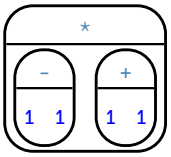
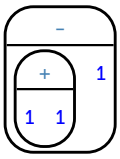
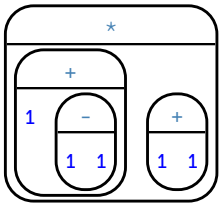
Completing Circles of Evaluation from Arithmetic Expressions

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

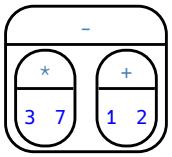
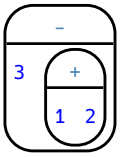
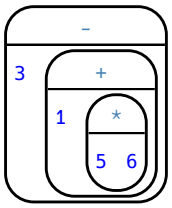
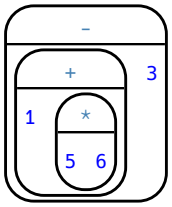
Matching Circles of Evaluation and Arithmetic Expressions

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

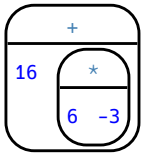
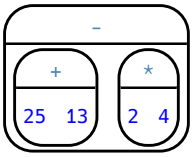
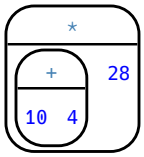
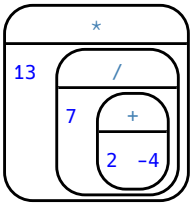
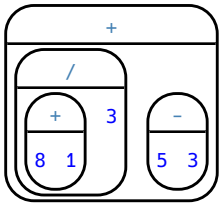
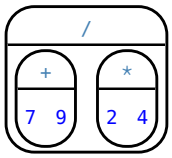
Translate Arithmetic to Circles of Evaluation & Code (Intro)

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code. **Teachers:** the answers below are shown with *all* parentheses included. There are also correct solutions in which the unecessary parens are left out.

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

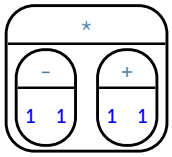
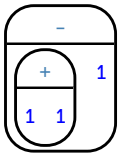
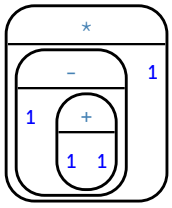
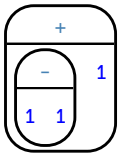
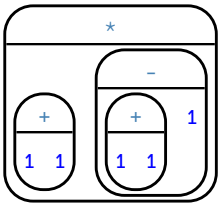
Completing Partial Code from Circles of Evaluation

For each Circle of Evaluation on the left, finish the Code on the right by filling in the blanks. **Teachers:** the answers below are shown with *all* parentheses included. There are also correct solutions in which the unecessary parens are left out.

	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))`

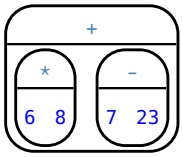
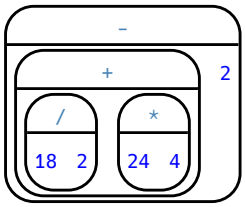
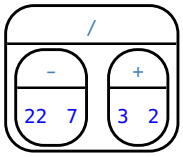
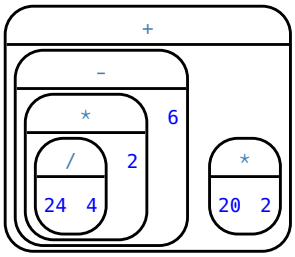
Matching Circles of Evaluation & Code

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)`

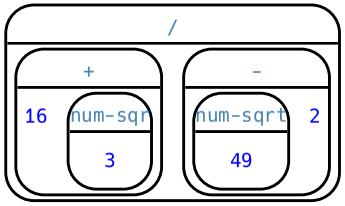
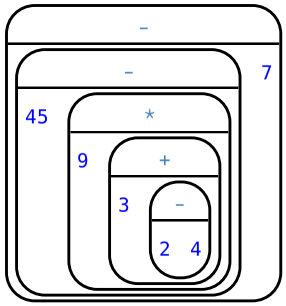
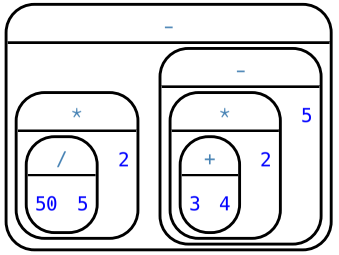
Translate Arithmetic to Circles of Evaluation & Code 2

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code. **Teachers:** the answers below are shown with *all* parentheses included. There are also correct solutions in which the unecessary parens are left out.

	Arithmetic	Circle of Evaluation	Code
1	$6 \times 8 + (7 - 23)$		`(6 * 8) + (7 - 23)`
2	$18 \div 2 + 24 \times 4 - 2$		`((18 / 2) + (24 * 4)) - 2`
3	$22 - 7 \div 3 + 2$		`(22 - 7) / (3 + 2)`
4	$24 \div 4 \times 2 - 6 + 20 \times 2$		`(((24 / 4) * 2) - 6) + (20 * 2)`

Arithmetic Expressions to Circles of Evaluation & Code - Challenge

Translate each of the arithmetic expressions below into Circles of Evaluation, then translate them to Code. **Teachers:** the answers below are shown with *all* parentheses included. There are also correct solutions in which the unecessary parens are left out.

	Arithmetic	Circle of Evaluation	Code
1	$\frac{16+3^2}{\sqrt{49}-2}$		`((16 + num-sqr(3)) / (num-sqrt(49) - 2))`
2	$45 - 9 \times (3 + (2 - 4)) - 7$		`((45 - (9 * (3 + (2 - 4)))) - 7)`
3	$50 \div 5 \times 2 - ((3 + 4) \times 2 - 5)$		`(((50 / 5) * 2) - (((3 + 4) * 2) - 5))`

Introduction to Programming

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 re-read everything in the Definitions Area and erase anything that was typed into the Interactions Area.

Data Types

Programming languages involve different ***data types***, such as Numbers, Strings, Booleans, and even Images.

- Numbers are values like `1`, `0.4`, `1/3`, and `-8261.003`.
  - Numbers are *usually* used for quantitative data and other values are *usually* used as categorical data.
  - In Pyret, any decimal *must* start with a 0. For example, `0.22` is valid, but `.22` is not.
- Strings are values like `"Emma"`, `"Rosanna"`, `"Jen and Ed"`, or even `"08/28/1980"`.
  - All strings *must* be surrounded in quotation marks.
- Booleans are either `true` or `false`.

All values evaluate to themselves. The program `42` will evaluate to `42`, the String `"Hello"` will evaluate to `"Hello"`, and the Boolean `false` will evaluate to `false`.

Operators

Operators (like `+`, `-`, `*`, `<`, etc.) work the same way in Pyret that they do in math.

- Operators are written between values, for example: `4 + 2`.
- In Pyret, operators must always have a space around them. `4 + 2` is valid, but `4+2` is not.
- If an expression has different operators, parentheses must be used to show order of operations. `4 + 2 + 6` and `4 + (2 * 6)` are valid, but `4 + 2 * 6` is not.

Applying Functions

Applying functions works much the way it does in math. Every function has a name, takes some inputs, and produces some output. The function name is written first, followed by a list of ***arguments*** in parentheses.

- In math this could look like $f(5)$ or $g(10, 4)$.
- In Pyret, these examples would be written as `f(5)` and `g(10, 4)`.
- Applying a function to make images would look like `star(50, "solid", "red")`.
- There are many other functions, for example `num-sqr`, `num-sqrt`, `triangle`, `square`, `string-repeat`, etc.

Functions have ***contracts***, which help explain how a function should be used. Every contract has three parts:

- The *Name* of the function - literally, what it's called.
- The *Domain* of the function - what *types of values* the function consumes, and in what order.
- The *Range* of the function - what *type of value* the function produces.

Numbers and Strings

Make sure you've loaded the code.pyret.org, (CPO) editor, clicked "Run", and are working in the ***Interactions Area***.

Numbers

1) Try typing `42` into the Interactions Area and hitting "Enter". What is the largest number the editor can handle?

I didn't find a number that was too big!

2) Try typing `0.5`. Then try typing `.5`. Then try clicking on the answer. Experiment with other decimals. Explain what you understand about how decimals work in this programming language.

In pyret, decimals need to be typed with a zero in front of the decimal point.

3) What happens if you try a fraction like `1/3`?

The editor will return the decimal equivalent. It even does repeating decimals!

4) Try writing negative integers, fractions and decimals.

Strings

String values are always in quotes.

5) Is `42` the same as `"42"`? Why or why not? Write your answer below:

No. 42 is a number and "42" is a string.

6) Try typing your name *(in quotes!)*.

7) Try typing a sentence like "I'm excited to learn to code!" *(in quotes!)*.

8) Try typing your name with the opening quote, but *without the closing quote.* Read the error message!

9) Now try typing your name *without any quotes.* Read the error message!

10) Explain what you understand about how strings work in this programming language.

Strings require quotes.

Operators

11) Just like math, Pyret has ***operators*** like `+`, `-`, `*` and `/`. Try typing in `4 + 2`, and then `4+2` (without the spaces). What can you conclude from this?

Operators require spaces.

12) Type in the following expressions, one at a time: `4 + 2 + 6`, `4 + 2 * 6`, `4 + (2 * 6)`. What do you notice?

If you use more than one operator you have to insert parentheses to tell the computer which operation to apply first.

13) Try typing in `4 + "cat"`, and then `"dog" + "cat"`. What can you conclude from this?

You can combine two strings, but you can't combine a number and a string.

Booleans

Boolean-producing expressions are yes-or-no questions and will always evaluate to either `true` ("yes") or `false` ("no"). What will each of the expressions below evaluate to? *Write down your prediction in the blanks provided and then type the code into the interactions area to see what it returns.*

	Prediction:	Computer Returns:		Prediction:	Computer Returns:
1) `3 <= 4`		*true*	2) `"a" > "b"`		*false*
3) `3 == 2`		*false*	4) `"a" < "b"`		*true*
5) `2 < 4`		*true*	6) `"a" == "b"`		*false*
7) `5 >= 5`		*true*	8) `"a" <> "a"`		*false*
9) `4 >= 6`		*false*	10) `"a" >= "a"`		*true*
11) `3 <> 3`		*false*	12) `"a" <> "b"`		*true*

13) In your own words, describe what `<` does.

checks whether the first number is smaller than the second number, returns true if it is and false if it isn't

14) In your own words, describe what `>=` does.

checks whether the first number is greater than or equal to the second number, returns true if it is and false if it isn't

15) In your own words, describe what `<>` does.

checks whether the to numbers are unequal, returns true if they're unequal and false if they're equal

	Prediction:	Computer Returns:
16) `string-contains("catnap", "cat")`		*true*
17) `string-contains("cat", "catnap")`		*false*

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

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

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

21) How many **Booleans** are there in the entire universe? *two*

Applying Functions

Type this line of code into the interactions area and hit "Enter":

```
triangle(50, "solid", "red")
```

1	What is the name of this function?	*triangle*
2	What did the expression evaluate to?	*an image of a red triangle*
3	How many arguments does `triangle` expect?	*3*
4	What data type does the `triangle` function produce? (Numbers? Strings? Booleans?)	*image*

Catching Bugs

The following lines of code are all BUGGY! Read the code and the error messages to identify the mistake.

5) `triangle(20, "solid" "red")`

Pyret didn't understand your program around
triangle(20, "solid" **"red"**)

Can you spot the mistake? *There is a missing comma after solid.*

6) `triangle(20, "solid")`

This application expression errored:
triangle (*20* , *"solid"*)
2 arguments were passed to the **operator** . The **operator** evaluated to a function accepting 3 parameters. An application expression expects the number of parameters and *arguments* to be the same.

Can you spot the mistake? *The color is undefined.*

7) `triangle(20, 10, "solid", "red")`

This application expression errored:
triangle (20, 10, "solid", *"red"*)`
4 arguments were passed to the **operator** . The **operator** evaluated to a function accepting 3 parameters. An application expression expects the number of parameters and *arguments* to be the same.

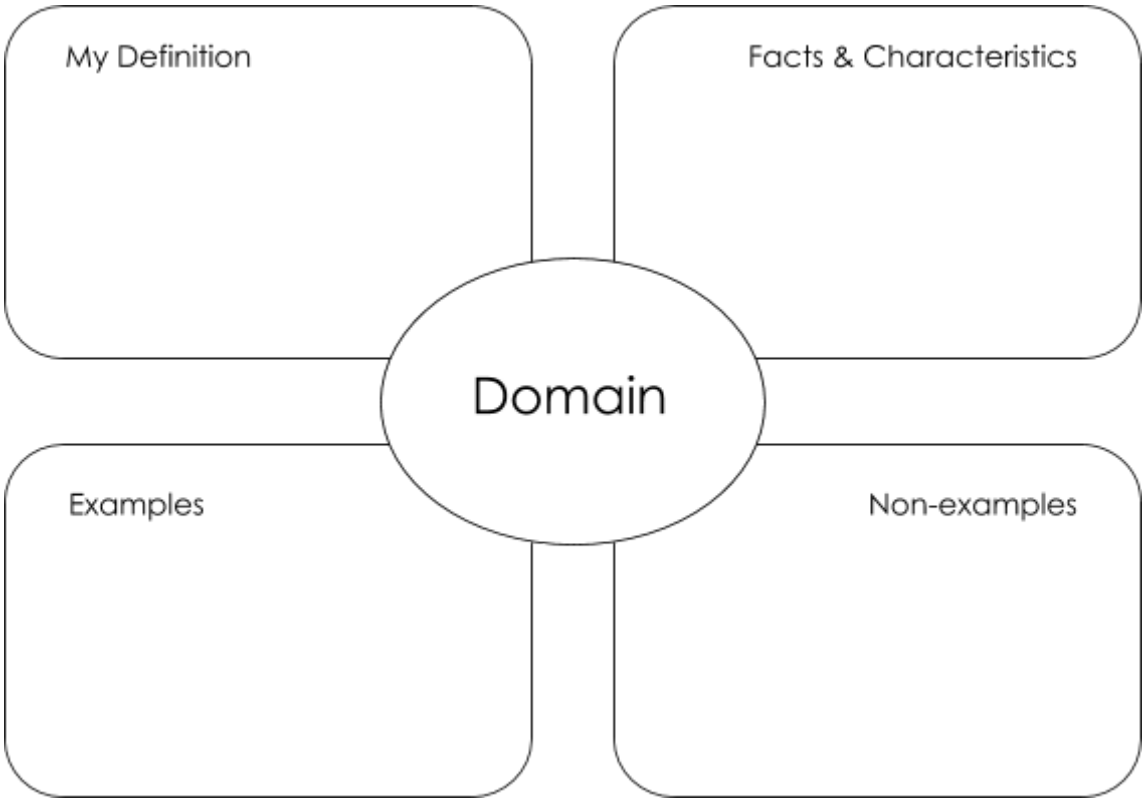
Can you spot the mistake? *There are too many numbers.*

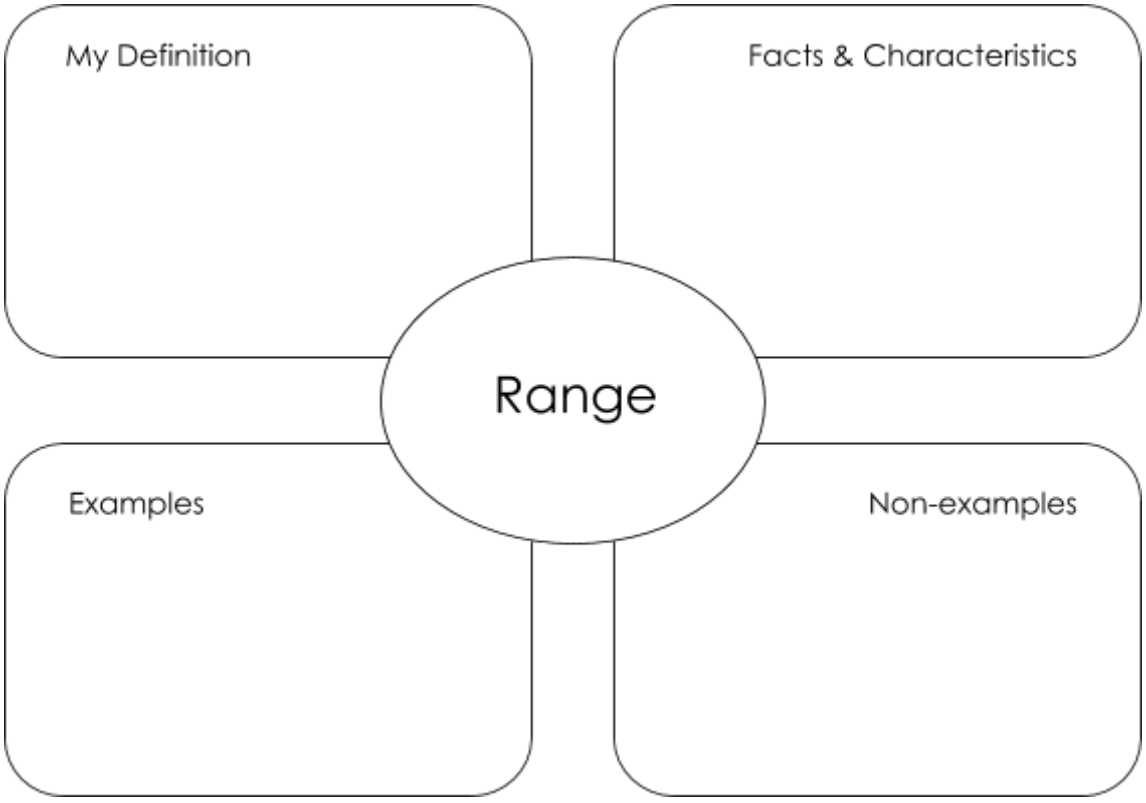
8) `triangle (20, "solid", "red")`

Pyret thinks this code is probably a function call:
triangle *(20, "solid", "red")*
Function calls must not have space between the **function expression** and the *arguments* .

Can you spot the mistake? *There shouldn't be a space between triangle and the parentheses.*

Domain and Range





Practicing Contracts: Domain & Range

Consider the following contract:

```
is-beach-weather :: Number, String -> Boolean
```

1) What is the **Name** of this function? *is-beach-weather*
2) How many arguments are in this function's **Domain**? *2*
3) What is the **type** of this function's **first argument**? *Number*
4) What is the **type** of this function's **second argument**? *String*
5) What is the **Range** of this function? *Boolean*

6) Circle the expression below that shows the correct application of this function, based on its contract.

A. `is-beach-weather(70, 90)`

B. `is-beach-weather(80, 100, "cloudy")`

C. `is-beach-weather("sunny", 90)`

D. * `is-beach-weather(90, "stormy weather")`

Consider the following contract:

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

7) What is the **Name** of this function? *cylinder*
8) How may arguments are in this function's **Domain**? *3*
9) What is the **type** of this function's **first argument**? *Number*
10) What is the **type** of this function's **second argument**? *Number*
11) What is the **type** of this function's **third argument**? *String*
12) What is the **Range** of this function? *Image*

13) Circle the expression below that shows the correct application of this function, based on its contract.

A. `cylinder("red", 10, 60)`

B. `cylinder(30, "green")`

C. * `cylinder(10, 25, "blue")`

D. `cylinder(14, "orange", 25)`

Matching Expressions and Contracts

Match the contract (left) with the expression described by the function being used (right).

Contract		Expression
`# make-id :: String, Number -> Image`	**1** -B	**A** `make-id("Savannah", "Lopez", 32)`
`# make-id :: String, Number, String -> Image`	**2** -C	**B** `make-id("Pilar", 17)`
`# make-id :: String -> Image`	**3** -E	**C** `make-id("Akemi", 39, "red")`
`# make-id :: String, String -> Image`	**4** -D	**D** `make-id("Raïssa", "McCracken")`
`# make-id :: String, String, Number -> Image`	**5** -A	**E** `make-id("von Einsiedel")`

Contract		Expression
`# is-capital :: String, String -> Boolean`	**6** -C	**A** `show-pop("Juneau", "AK", 31848)`
`# is-capital :: String, String, String -> Boolean`	**7** -E	**B** `show-pop("San Juan", 395426)`
`# show-pop :: String, Number -> Image`	**8** -B	**C** `is-capital("Accra", "Ghana")`
`# show-pop :: String, String, Number -> Image`	**9** -A	**D** `show-pop(3751351, "Oklahoma")`
`# show-pop :: Number, String -> Number`	**10** -D	**E** `is-capital("Albany", "NY", "USA")`

Using Contracts

Use the contracts to write expressions to generate images similar to those pictured.

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

	`ellipse(50, 150, "outline", "black")`
	`ellipse(150, 50, "solid", "black")`
What changes with the first number?	*width*
What about the shape changes with the second Number?	*height*
Write an expression using `ellipse` to produce a circle.	`ellipse(50, 50, "solid", "black")`

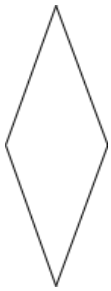
`regular-polygon :: Number, Number, String, String -> Image`

	`regular-polygon(50, 5, "solid", "black")`
	`regular-polygon(25, 8, "outline", "black")`
What changes with the first Number?	*the length of each side*
What about the shape changes with the second Number?	*the number of sides*
Use `regular-polygon` to write an expression for a square!	`regular-polygon(50, 4, "solid", "black")`
How would you describe a **regular polygon** to a friend?	*a closed shape made of equal-length straight lines, all joined at the same angle*

Using Contracts (continued)

Use the contracts to write expressions to generate images similar to those pictured.

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

	`rhombus(100, 140, "solid", "black")`
	`rhombus(100, 40, "outline", "black")`
Write an expression for a square (rotated) using `rhombus`!	`rhombus(100, 90, "solid", "black")`
What variable changes with the first Number?	*The length of each side*
What variable changes with the second Number?	*The top and bottom angle of the rhombus*

Triangle Contracts

1) What kind of triangle does the `triangle` function produce? *equilateral*

There are lots of other kinds of triangles! And Pyret has lots of other functions that make triangles!

```
triangle :: (size:: Number, style :: String, color :: String) -> Image
right-triangle :: (base::Number, height::Number, style::String, color::String) -> Image
isosceles-triangle :: (leg::Number, angle::Number, style::String, color::String) -> Image
```

2) Why do you think `triangle` only needs one number, while `right-triangle` and `isosceles-triangle` need two numbers and `triangle-sas` needs three?

triangle makes equilateral triangles, so the angle is fixed at 60°. right-triangle needs to know the length of the two legs, and triangle/sas needs to know both sides AND an angle

3) Write `right-triangle` expressions for the images below. *One argument for each should be* `100` .

```
right-triangle(50, 100, "solid", "black")
```

```
right-triangle(100, 50, "solid", "black")
```

4) What do you think the numbers in `right-triangle` represent? *base & height*

5) Write `isosceles-triangle` expressions for the images below. *1 argument for each should be* `100` .

```
isosceles-triangle(100, 30, "solid", "black")
```

```
isosceles-triangle(100, 130, "solid", "black")
```

6) What do you think the numbers in `isosceles-triangle` represent?

The length of the sides, and the size of the angle between them

7) Write 2 expressions that would build **right-isosceles** triangles. Use `right-triangle` for one expression and `isosceles-triangle` for the other expression.

```
right-triangle(100, 100, "solid", "black")
```

```
isosceles-triangle(100, 90, "solid", "black")
```

Radial Star

```
radial-star :: (
   points :: Number,
   inner-radius :: Number,
   full-radius :: Number,
   style :: String,
   color :: String
  ) -> Image
```

Using the detailed contract above, match each image to the expression that describes it.

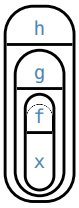
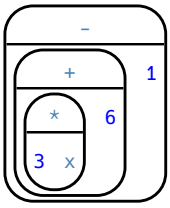
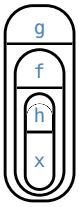
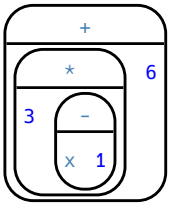
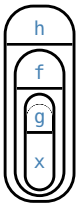
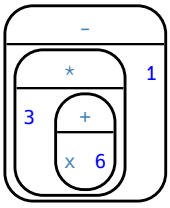
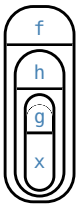
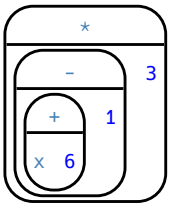
Image			Expression
	**1** -D	**A**	`radial-star(5, 50, 200, "solid", "black")`
	**2** -A	**B**	`radial-star(7, 100, 200, "solid", "black")`
	**3** -G	**C**	`radial-star(7, 100, 200, "outline", "black")`
	**4** -F	**D**	`radial-star(10, 150, 200, "solid", "black")`
	**5** -C	**E**	`radial-star(10, 20, 200, "solid", "black")`
	**6** -D	**F**	`radial-star(100, 20, 200, "solid", "black")`
	**7** -E	**G**	`radial-star(100, 100, 200, "outline", "black")`

What's on your mind?

Diagramming Function Composition

`f :: Number -> Number` `Consumes a number, multiplies by 3 to produce the result`	`g :: Number -> Number` `Consumes a number, adds six to produce the result`	`h :: Number -> Number` `Consumes a number, subtracts one to produce the result`
$f(x) = 3x$	$g(x) = x + 6$	$h(x) = x - 1$

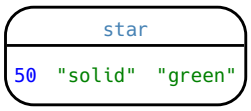
For each function composition diagrammed below, translate it into the equivalent Circle of Evaluation for Order of Operations. Then write expressions for *both* versions of the Circles of Evaluation, and evaluate them for $x = 4$. The first one has been completed for you.

Function Composition	Order of Operations	Translate & Evaluate
1) 		Composition: `h(g(f(x)))` Operations: `((3 * x) + 6) - 1` Evaluate for x = 4: $h(g(f(4))) = 17$
2) 		Composition: `g(f(h(x)))` Operations: `(3 * (x - 1)) + 6` Evaluate for x = 4: g(f(h(4))) = 15
3) 		Composition: `h(f(g(x)))` Operations: `(3 * (x + 6)) - 1` Evaluate for x = 4: h(f(g(4))) = 29
4) 		Composition: `f(h(g(x)))` Operations: `((x + 6) - 1) * 3` Evaluate for x = 4: f(h(g(4))) = 27

Function Composition — Green Star

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

Circle of Evaluation:

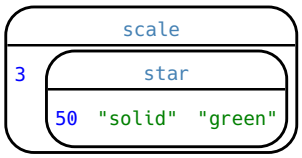


Code: `star(50, "solid", "green")`

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`)

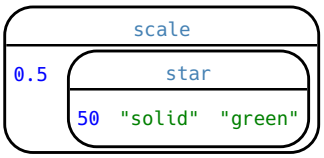
Circle of Evaluation:



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

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

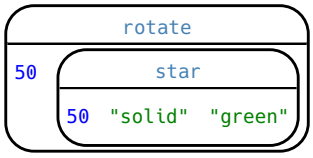
Circle of Evaluation:



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

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

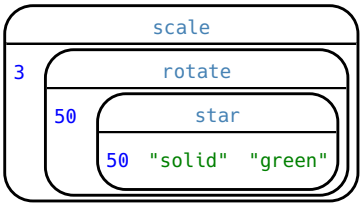
Circle of Evaluation:



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

Function Composition — Your Name

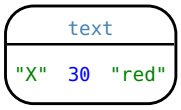
You'll be investigating these functions with your partner:

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

```
# frame :: Image -> Image
# above :: Image, Image -> Image
# beside :: Image, Image -> Image
```

1) In the editor, write the code to make an image of your name in big letters in a color of your choosing using `text`. Then draw the Circle of Evaluation and write the Code that will create the image.

Circle of Evaluation:

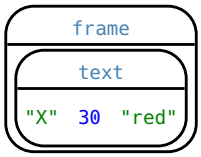


Code: `text("X", 30, "red")`

Using the "image of your name" described above as the **original**, draw the Circles of Evaluation and write the Code for each exercise below. Test your ideas in the editor to make sure they work.

2 The framed "image of your name".

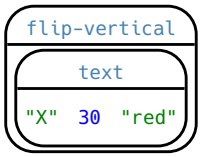
Circle of Evaluation:



Code: `frame(text("X", 30, "red"))`

3 The "image of your name" flipped vertically.

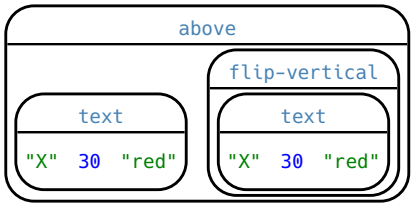
Circle of Evaluation:



Code: `flip-vertical(text("X", 30, "red"))`

4 The "image of your name" above "the image of your name" flipped vertically.

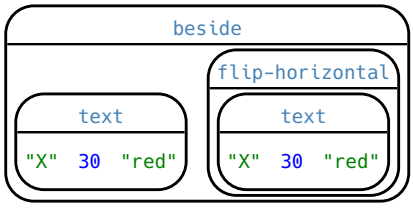
Circle of Evaluation:



Code: `above(text("X", 30, "red"), flip-vertical(text("X", 30, "red")))`

5 The "image of your name" flipped horizontally beside "the image of your name".

Circle of Evaluation:

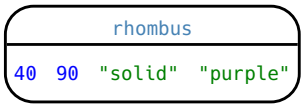


Code: `beside(text("X", 30, "red"), flip-horizontal(text("X", 30, "red")))`

Function Composition — scale-xy

You'll be investigating these two functions with your partner:

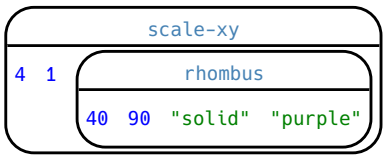
```
# scale-xy :: Number, Number, Image -> Image        # overlay :: Image, Images -> Image
```

The Image:	Circle of Evaluation:	Code:
		`rhombus(40, 90, "solid", "purple")`

Starting with the image described above, write the Circles of Evaluation and Code for each exercise below. Be sure to test your code in the editor!

1 A purple rhombus that is stretched 4 times as wide.

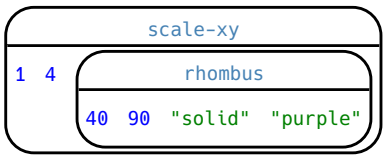
Circle of Evaluation:



Code: `scale-xy(4, 1, rhombus(40, 90, "solid", "purple"))`

2 A purple rhombus that is stretched 4 times as tall

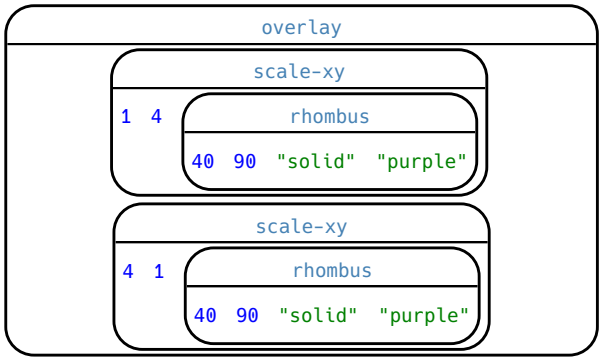
Circle of Evaluation:



Code: `scale-xy(1, 4, rhombus(40, 90, "solid", "purple"))`

3 The tall rhombus overlayed on the wide rhombus.

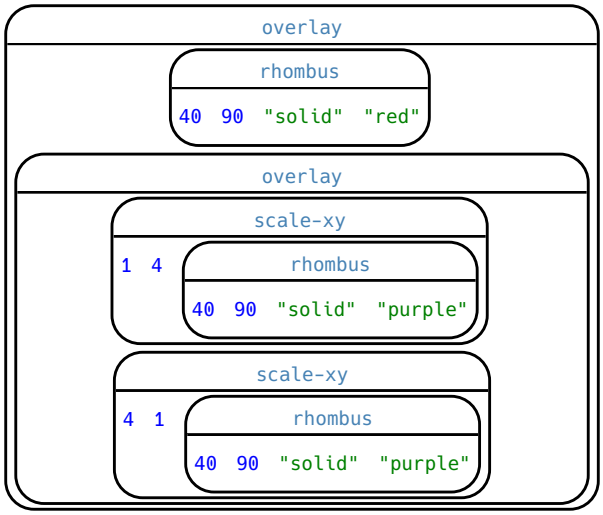
Circle of Evaluation:



Code: `overlay(scale-xy(1, 4, rhombus(40, 90, "solid", "purple")), scale-xy(4, 1, rhombus(40, 90, "solid", "purple")))`

★: Overlay a red rhombus onto the last image you made.

Circle of Evaluation:



Code: `overlay(rhombus(40, 90, "solid", "red"), overlay(scale-xy(1, 4, rhombus(40, 90, "solid", "purple")), scale-xy(4, 1, rhombus(40, 90, "solid", "purple"))))`

More than one way to Compose an Image!

Read through these 4 expressions and try to picture the images they are composing. If you're not sure what they'll look like, type them into the interactions area of your editor and see if you can figure out how the code connects to the image.

```
beside(rectangle(200, 100, "solid", "black"), square(100, "solid", "black"))
scale-xy(1, 2, square(100, "solid", "black"))
scale(2, rectangle(100, 100, "solid", "black"))
above(
 rectangle(100, 50, "solid", "black"),
 above(
  rectangle(200, 100,  "solid", "black"),
  rectangle(100, 50, "solid", "black")))
```

For each image below, identify 2 expressions that could be used to compose it. *The bank of expressions at the top of the page includes one possible option for each image.*

1		• `rotate(90, rectangle(200, 100, "solid", "black"))` • `scale-xy(1, 2, square(100, "solid", "black"))` • *answers may vary*
2		• `above(rectangle(200, 100, "solid", "black"), rectangle(200, 100, "solid", "black"))` • `scale(2, rectangle(100, 100, "solid", "black"))` • *answers may vary*
3		• `scale(0.5, rectangle(600, 200, "solid", "black"))` • `beside(rectangle(200, 100, "solid", "black"), square(100, "solid", "black"))` • *answers may vary*

Defining Values

In math, we use **values** like $-98.1, 2/3$ amd 42. In math, we also use **expressions** like 1×3, $\sqrt{16}$, and $5 - 2$. These evaluate to results, and typing any of them in as code produces some answer.

Math also has **definitions**. These are different from values and expressions, because they *they do not produce results*. Instead, they simply create names for values, so that those names can be re-used to make the Math simpler and more efficient.

Definitions always have both a name and an expression. The name goes on the left and the value-producing expression goes on the right, separated by an equals sign:

$x = 4$

$y = 9 + x$

The name is defined to be the result of evaluating the expression. Using the above examples, we get "x is defined to be 4, and y is defined to be 13". **Important: there is no "answer" to a definition**, and typing in a definition as code will produce no result.

Notice that *definitions can refer to previous definitions*. In the example above, the definition of `y` refers to `x`. But `x`, on the other hand, *cannot* refer to `y`. Once a value has been defined, it can be used in later expressions.

In Pyret, these definitions are written the *exact same way*:
Try typing these definitions into the Definitions Area on the left, clicking "Run", and then *using* them in the Interactions Area on the right.

```
x = 4
y = 9 + x
```

Just like in math, definitions in our programming language can only refer to previously-defined values.

Here are a few more value definitions. Feel free to type them in, and make sure you understand them.

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

Defining Values - Explore

Open the Defining Values Starter File and click run.

1) What do you notice?

2) What do you wonder?

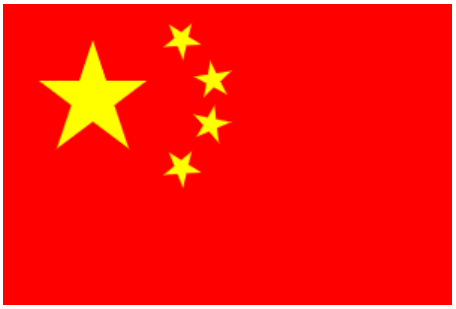
Look at the expressions listed below. Think about what you expect each of them to produce. Then, test them out one at a time in the Interactions Area.

- `x`
- `x + 5`
- `y - 9`
- `x * y`
- `z`
- `t`
- `gold-star`
- `my-name`
- `swamp`
- `c`

3) What have you learned about defining values?

4) Define at least 2 more variables in the definitions area, click run and test them out. Once you know they're working, record the code you used below.

Defining Values - Chinese Flag



1) What image do you see repeated in the flag? *small yellow stars*

2) Highlight or circle all instances of the structure that makes the repeated image in the code below.

3) In the code below, highlight or circle all instances of the expression for that image.

students should highlight or circle: `star(15, "solid", "yellow")`

```
put-image(
  rotate(40, star(15, "solid", "yellow")),
  120, 175,
  put-image(
    rotate(80, star(15, "solid", "yellow")),
    140, 150,
    put-image(
      rotate(60, star(15, "solid", "yellow")),
      140, 120,
      put-image(
        rotate(40, star(15, "solid", "yellow")),
        120, 90,
        put-image(scale(3, star(15, "solid", "yellow")),
          60, 140,
          rectangle(300, 200, "solid", "red"))))))
```

4) Write the code to define a value for the repeated expression.

```
yellow-star = star(15, "solid", "yellow")
```

5) Open the Chinese flag starter file (Pyret) and click Run.

Then type `china` into the interactions area and click **Enter**.

6) **Save a copy** of the file, and simplify the flag code using the value you defined. Click Run, and confirm that you still get the same image as the original.

7) Now change the color of all of the stars to black, in both files. Then change the size of the stars.

8) Why is it helpful to define values for repeated images?

Challenge:

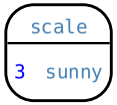
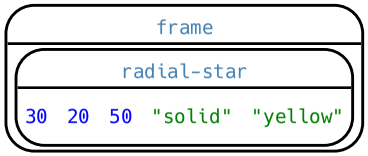
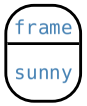
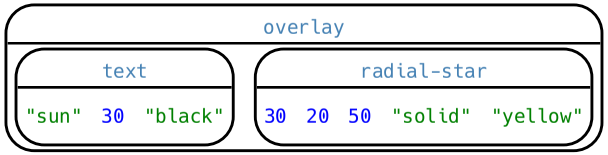
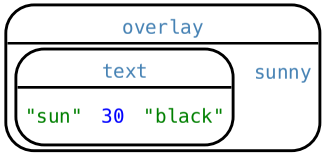
- This file uses a function we haven't seen before! What is it? *put-image*
- Can you figure out its contract? *Hint: Focus on the last instance of the function.*

put-image :: Image, Number, Number, Image -> Image

Why Define Values?

1) Complete the table using the first row as an example.

2) Write the code to define the value of `sunny`. `sunny = radial-star(30, 20, 50, "solid", "yellow")`

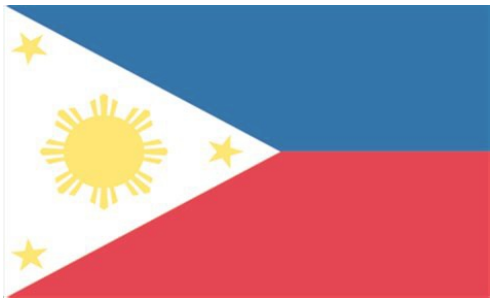
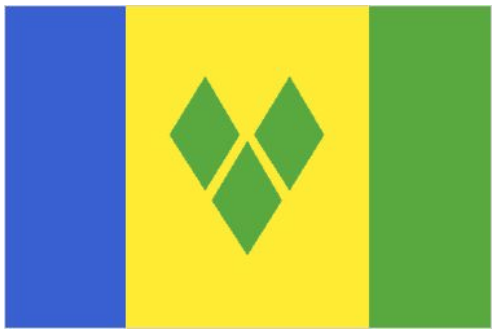
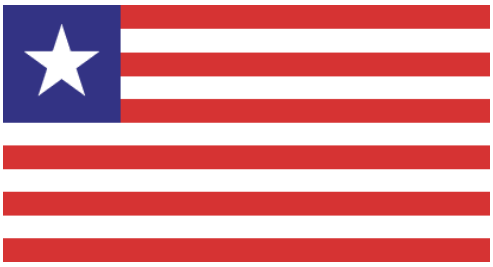
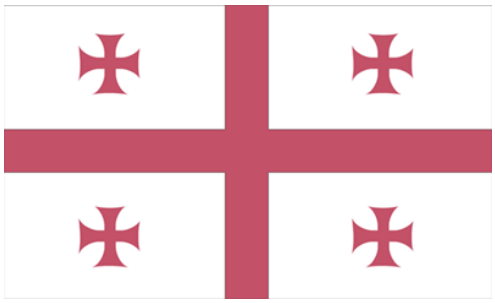
Original Circle of Evaluation & Code	→	Use the *defined value* `sunny` to simplify!
	→	
Code: `scale(3, radial-star(30, 20, 50, "solid", "yellow"))`	→	Code: `scale(3, sunny)`
	→	
Code: `frame(radial-star(30, 20, 50, "solid", "yellow"))`	→	Code: `frame(sunny)`
	→	
Code: `overlay(text("sun", 30, "black"), radial-star(30, 20, 50, "solid", "yellow"))`	→	Code: `overlay(text("sun", 30, "black"), sunny)`

3) Test your code in the editor and make sure it produces what you would expect it to.

Which Value(s) Would it Make Sense to Define?

For each of the images below, identify which element(s) you would want to define before writing code to compose the image.

Hint: what gets repeated?

Philippines	St. Vincent & the Grenadines
	
yellow star	*green diamonds*
Liberia	**Republic of Georgia**
	
red rectangles	*red crosses*
Quebec	**South Korea**
	
fleur de lis	*black rectangles*

Writing Code using Defined Values

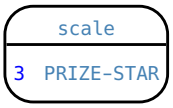
1) On the line below, **write the Code** to define `PRIZE-STAR` as a pink, outline star of size 65.

```
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 three times the size of the original (using `scale`)

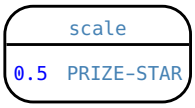
Circle of Evaluation:



Code: `scale(3, PRIZE-STAR)`

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

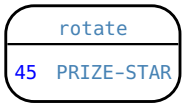
Circle of Evaluation:



Code: `scale(0.5, PRIZE-STAR)`

4 The outline of a pink star that is rotated 45 degrees *(It should be the same size as the original.)*

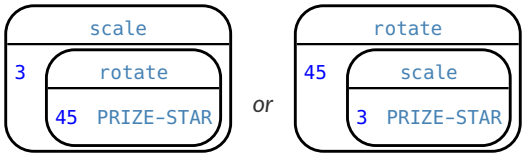
Circle of Evaluation:



Code: `rotate(45, PRIZE-STAR)`

5 The outline of a pink star that is three times as big as the original **and** has been rotated 45 degrees

Circle of Evaluation:



Code: `scale(3, rotate(45, PRIZE-STAR))` *or* `rotate(45, scale(3, PRIZE-STAR))`

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

Estimating Coordinates

Think of the background image as a sheet of graph paper with the origin (0,0) in the bottom left corner.

The numbers in `put-image` specify a point on that graph paper, where the center of the top image should be placed.

The width of the rectangle is 300 and the height is 200.

Estimate: What coordinates for the `dot` would create each of the following images?

A

put-image(dot, 0 , 0 background)

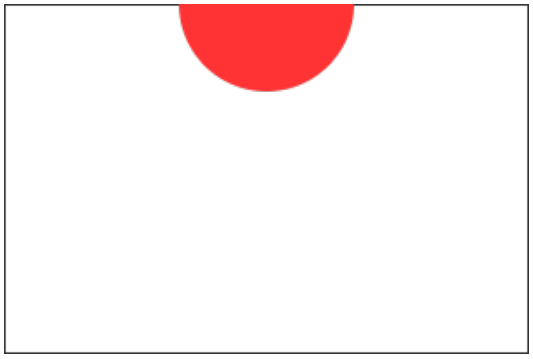
B

put-image(dot, 0 , 100 background)

C

put-image(dot, 300 , 200 background)

D



put-image(dot, 150 , 200 background)

Decomposing Flags

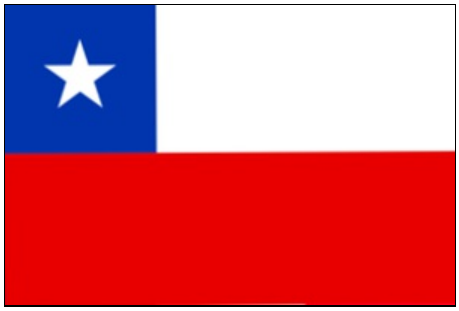
Each of the flags below is shown with their width and height. Identify the shapes that make up each flag. Use the flag's dimensions to estimate the dimensions of the different shapes. Then estimate the x and y coordinates for the point at which the center of each shape should be located on the flag. *Hint: The bottom left corner of each flag is at (0,0) and the top right corner is given by the flags dimensions.*

Each of these images can be composed multiple ways. The solutions below represent one possibility.

Cameroon (450 x 300)

shape:	color:	width:	height:	x	y
rectangle	*green*	*150*	*300*	*75*	*150*
rectangle	*red*	*150*	*300*	*225*	*150*
star	*yellow*	*30*	*n/a*	*225*	*150*
rectangle	*yellow*	*450*	*300*	*n/a*	*n/a*

Chile (420 x 280)

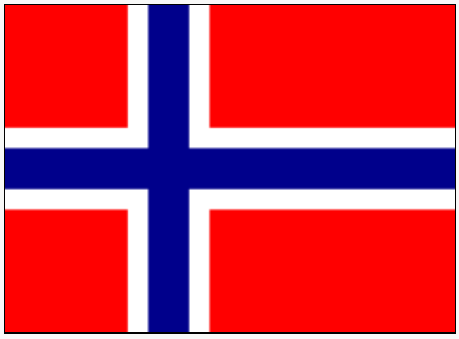


shape:	color:	width:	height:	x	y
square	*blue*	*140*	*140*	*70*	*210*
rectangle	*white*	*280*	*140*	*350*	*210*
star	*white*	*30*	*n/a*	*70*	*210*
rectangle	*red*	*420*	*280*	*n/a*	*n/a*

Panama (300 x 200)

shape:	color:	width:	height:	x	y
rectangle	*blue*	*150*	*100*	*75*	*50*
rectangle	*red*	*150*	*100*	*225*	*150*
star	*blue*	*30*	*n/a*	*75*	*150*
star	*red*	*30*	*n/a*	*225*	*50*
rectangle	*white*	*300*	*200*	*n/a*	*n/a*

Norway (330 x 240)



shape:	color:	width:	height:	x	y
rectangle	*dark blue*	*330*	*30*	*165*	*120*
rectangle	*dark blue*	*30*	*240*	*120*	*120*
rectangle	*white*	*330*	*60*	*165*	*120*
rectangle	*white*	*60*	*240*	*120*	*120*
rectangle	*red*	*330*	*240*	*n/a*	*n/a*

Notice and Wonder

As you investigate the Game Starter 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?

Defining Functions

Functions can be viewed in *multiple representations*. You already know one of them: ***Contracts***, which specify the Name, Domain, and Range of a function. Contracts are a way of thinking of functions as a *mapping* between one set of data and another. For example, a mapping from Numbers to Strings:

```
f :: Number -> String
```

Another way to view functions is with ***Examples***. Examples are essentially input-output tables, showing what the function would do for a specific input:

In our programming langauge, we focus on the last two columns and write them as code:

```
examples:
  f(1) is 1 + 2
  f(2) is 2 + 2
  f(3) is 3 + 2
  f(4) is 4 + 2
end
```

Finally, we write a formal **function definition** ourselves. The pattern in the Examples becomes *abstract* (or "general"), replacing the inputs with ***variables***. In the example below, the same definition is written in both math and code:

$f(x) = x + 2$

```
fun f(x): x + 2 end
```

Look for connections between these three representations!

- The function name is always the same, whether looking at the Contract, Examples, or Definition.
- The number of inputs in the Examples is always the same as the number of types in the Domain, which is always the same as the number of variables in the Definition.
- The "what the function does" pattern in the Examples is almost the same in the Definition, but with specific inputs replaced by variables.

Matching Examples and Definitions (Math)

Look at each set of examples on the left and circle what is changing from one example to the next.

Then, *match* the examples on the left to the definitions on the right.

Examples:		Functions:	
x	$f(x)$		
1	2×1		
2	2×2		
3	2×3		
		1 -B	**A** $f(x) = x - 3$
x	$f(x)$		
15	$15 - 3$		
25	$25 - 3$		
35	$35 - 3$		
		2 -A	**B** $f(x) = 2x$
x	$f(x)$		
10	$10 + 2$		
15	$15 + 2$		
20	$20 + 2$		
		3 -E	**C** $f(x) = 2x + 1$
x	$f(x)$		
0	$3(0) - 2$		
1	$3(1) - 2$		
2	$3(2) - 2$		
		4 -D	**D** $f(x) = 3x - 2$
x	$f(x)$		
10	$2(10) + 1$		
20	$2(20) + 1$		
30	$2(30) + 1$		
		5 -C	**E** $f(x) = x + 2$

Matching Examples and Function Definitions

Highlight the variables in `gt` and label them with the word "size".

```
examples:
  gt(20) is
    triangle(20, "solid", "green")
  gt(45) is
    triangle(45, "solid", "green")
end
```

```
fun gt(size): triangle(size, "solid", "green") end
```

Highlight and label the variables in the example lists below. Then, using `gt` as a model, match the examples to their corresponding function definitions.

Examples		Definition

`examples:`
`  f("solid") is`
`  circle(8, "solid", "red")`
`  f("outline") is`
`  circle(8, "outline", "red")`
`end`
	1 - D	A `fun f(s): star(s, "outline", "red") end`

`examples:`
`  f(2) is 2 + 2`
`  f(4) is 4 + 4`
`  f(5) is 5 + 5`
`end`
	2 - B	B `fun f(num): num + num end`

`examples:`
`  f("red") is circle(7, "solid", "red")`
`  f("teal") is`
`  circle(7, "solid", "teal")`
`end`
	3 - E	C `fun f(c): star(9, "solid", c) end`

`examples:`
`  f("red") is star(9, "solid", "red")`
`  f("grey") is star(9, "solid", "grey")`
`  f("pink") is star(9, "solid", "pink")`
`end`
	4 - C	D `fun f(s): circle(8, s, "red") end`

`examples:`
`  f(3) is star(3, "outline", "red")`
`  f(8) is star(8, "outline", "red")`
`end`
	5 - A	E `fun f(c): circle(7, "solid", c) end`

Matching Examples and Contracts

Match each set of examples (left) with the contract that best describes it(right).

Examples			Contract

`examples:`
`  f(5) is 5 / 2`
`  f(9) is 9 / 2`
`  f(24) is 24 / 2`
`end`
	1-A	A	`# f :: Number -> Number`

`examples:`
`  f(1) is`
`  rectangle(1, 1, "outline", "red")`
`  f(6) is`
`  rectangle(6, 6, "outline", "red")`
`end`
	2-C	B	`# f :: String -> Image`

`examples:`
`  f("pink", 5) is`
`  star(5, "solid", "pink")`
`  f("blue", 8) is`
`  star(8, "solid", "blue")`
`end`
	3-E	C	`# f :: Number -> Image`

`examples:`
`  f("Hi!") is text("Hi!", 50, "red")`
`  f("Ciao!") is text("Ciao!", 50, "red")`
`end`
	4-B	D	`# f :: Number, String -> Image`

`examples:`
`  f(5, "outline") is`
`  star(5, "outline", "yellow")`
`  f(5, "solid") is`
`  star(5, "solid", "yellow")`
`end`
	5-D	E	`# f :: String, Number -> Image`

Contracts, Examples & Definitions

gt

Directions: Define a function called `gt`, which makes solid green triangles of whatever size we want.

Every contract has three parts...

```
# gt :: Number -> Image
```
function name / *domain* / *range*

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

```
examples:
  gt ( 10 ) is triangle(10, "solid", "green")
  gt ( 20 ) is triangle(20, "solid", "green")
end
```
function name / *input(s)* / *what the function produces*

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

```
fun gt ( size ):
  triangle(size, "solid", "green")
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

bc

Directions: Define a function called `bc`, which makes solid blue circles of whatever radius we want.

Every contract has three parts...

```
# bc :: Number -> Image
```
function name / *domain* / *range*

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

```
examples:
  bc ( 10 ) is circle(10, "solid", "blue")
  bc ( 20 ) is circle(20, "solid", "blue")
end
```
function name / *input(s)* / *what the function produces*

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

```
fun bc ( radius ):
  circle(radius, "solid", "blue")
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

What's on your mind?

Solving Word Problems

Being able to see functions as Contracts, Examples or Definitions is like having three powerful tools. These representations can be used together to solve word problems!

1) When reading a word problem, the first step is to figure out the **Contract** for the function you want to build. Remember, a Contract must include the Name, Domain and Range for the function!

2) 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!

3) Next, we 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.

4) To finish the Examples, we circle the parts that are changing, and label them with a short **variable name** that explains what they do.

5) Finally, we 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!

Creating Contracts From Examples

Write the contracts used to create each of the following collections of examples.

1)
```
# big-triangle :: Number, String -> Image
examples:
  big-triangle(100, "red") is
    triangle(100, "solid", "red")
  big-triangle(200, "orange") is
    triangle(200, "solid", "orange")
end
```

2)
```
# purple-square :: Number -> Image
examples:
  purple-square(15) is
    rectangle(15, 15, "outline", "purple")
  purple-square(6) is
    rectangle(6, 6, "outline", "purple")
end
```

3)
```
# banner :: String -> Image
examples:
  banner("Game Today!") is
    text("Game Today!", 50, "red")
  banner("Go Team!") is
    text("Go Team!", 50, "red")
  banner("Exit") is
    text("Exit", 50, "red")
end
```

4)
```
# twinkle :: String, String -> Image
examples:
  twinkle("outline", "red") is
    star(5, "outline", "red")
  twinkle("solid", "pink") is
    star(5, "solid", "pink")
  twinkle("outline", "grey") is
    star(5, "outline", "grey")
end
```

5)
```
# half :: Number -> Number
examples:
  half(5) is 5 / 2
  half(8) is 8 / 2
  half(900) is 900 / 2
end
```

Writing Examples from Purpose Statements

We've provided contracts and purpose statements to describe two different functions. Write examples for each of those functions.

Contract and Purpose Statement

Every contract has three parts…

```
# upside-down:: Image -> Image
```
function name *domain* *range*

```
# Consumes an image, and flips it upside down by rotating it 180 degrees.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  upside-down (triangle(50, "solid", "yellow")) is
  rotate(180, triangle(50, "solid", "yellow"))
  upside-down (star(150, "solid", "blue")) is
  rotate(180, star(150, "solid", "blue"))
end
```
function name *input(s)* *what the function produces*

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

Contract and Purpose Statement

Every contract has three parts…

```
# product-squared:: Number, Number -> Number
```
function name *domain* *range*

```
# Consumes two numbers and squares their product
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  product-squared ( 5, 6 ) is num-sqr(5 * 6)
  product-squared ( 10, 17 ) is num-sqr(10 * 17)
end
```
function name *input(s)* *what the function produces*

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

Word Problem: rocket-height

Directions : A rocket blasts off, and is now traveling at a constant velocity of 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*

```
# Consumes 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
  rocket-height ( 3 ) is 7 * 3
end
```
function name — *input(s)* — *what the function produces*

Definition

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

```
fun rocket-height( seconds ):
  7 * seconds
end
```
function name — *variable(s)*

what the function does with those variable(s)

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:

The Design Recipe - Direct Variation

Directions: Write a function `wage`, that takes in a number of hours worked and returns the amount a worker will get paid if their rate is $10.25/hr.

Contract and Purpose Statement

Every contract has three parts…

```
# wage:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes 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:
  wage ( 0 ) is 0 * 10.25
  wage ( 30 ) is 30 * 10.25
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun wage( hours ):
  hours * 10.25
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

Directions: On average, people burn about 11 calories/minute riding a bike. Write a function `calories-burned` that takes in the number of minutes you bike and returns the number of calories burned. .

Contract and Purpose Statement

Every contract has three parts…

```
# calories-burned:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes minutes biked, and multiplies by 11 to produce the calories.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  calories-burned ( 10 ) is 10 * 11
  calories-burned ( 200 ) is 200 * 11
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun calories-burned( minutes ):
  minutes * 11
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

The Design Recipe (Practice 1)

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*

```
# Consumes 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 ( "It works" ) is text("It 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 `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*

```
# Consumes 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)
  num-cube ( 3 ) is 3 * (3 * 3)
end
```
function name — *input(s)* — *what the function produces*

Definition

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

```
fun num-cube( n ):
  n * (n * n)
end
```
function name — *variable(s)*

what the function does with those variable(s)

The Design Recipe (Practice 2)

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*

```
# Consumes a cost and the # of people splitting, and produces each person's share
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  split-tab ( 200, 10 ) is 200 / 10
  split-tab ( 500, 5 ) is 500 / 5
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun split-tab( cost, people ):
  cost / people
end
```
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…

```
# tip-calculator:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes cost of a meal, and multiplies by 0.15 to produce 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
  tip-calculator ( 35 ) is 0.15 * 35
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun tip-calculator( cost ):
  0.15 * cost
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

The Design Recipe (Practice 3)

Directions : The Swamp in the City Festival is ordering t-shirts. The production cost is $75 to set up the silk screen and $9 per shirt. Write a function `min-shirt-price` that takes in the number of shirts to be ordered, *n* , and returns the minimum amount the festival should charge for the shirts in order to break even. (Assume that they will sell all of the shirts.)

Contract and Purpose Statement

Every contract has three parts…

```
# min-shirt-price:: Number -> Number
```
function name *domain* *range*

```
# Consumes number of shirts, multiplies it by nine, adds 75, and then divides the sum by that number.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  min-shirt-price ( 10 ) is ((9 * 10) + 75) / 10
  min-shirt-price ( 100 ) is ((9 * 100) + 75) / 100
end
```
function name *input(s)* *what the function produces*

Definition

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

```
fun min-shirt-price( length, width ):
  ((9 * n) + 75) / n
end
```
function name *variable(s)* *what the function does with those variable(s)*

The Design Recipe (Slope/Intercept 1)

Directions : For his birthday, James' family decided to open a savings account for him. He started with $50 and committed to adding $10 a week from his afterschool job teaching basketball to kindergartners. Write a function `savings` that takes in the number of weeks since his birthday and calculates how much money he has saved.

Contract and Purpose Statement

Every contract has three parts…

```
# savings:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes number of weeks, multiplies by 10 and returns the sum of the product and 50.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
    savings ( 10 ) is (10 * 10) + 50
    savings ( 17 ) is (10 * 17) + 50
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun savings( weeks ):
  (10 * weeks) + 50
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

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 $45 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*

```
# Consumes a number of days and multiplies it by $45, takes in a number of miles and multiplies it by
  $0.15, then adds the two products to produce 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 * 45) + (600 * 0.15)
    moving ( 3, 1500 ) is (3 * 45) + (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 * 45) + (miles * 0.15)
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

The Design Recipe (Negative Slope/Intercept)

Directions : An Olympic pool holds 660,000 gallons of water. A fire hose can spray about 250 gallons per minute. Write a function `pool` that takes in the number of minutes that have passed and calculates how much water is still needed to fill it.

Contract and Purpose Statement

Every contract has three parts…

```
# pool:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes the number of minutes, multiplies them by 250 and subtracts the product from 660,000.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  pool ( 10 ) is 660000 - (10 * 250)
  pool ( 40 ) is 660000 - (40 * 250)
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun pool( n ):
  660000 - (minutes * 250)
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

Directions : The community arts fund awards a $1500 grant each month to support a new mural. They started with $50000 in their account. Write a function `funds-available` that takes in the number of months and calculates how much money they have left.

Contract and Purpose Statement

Every contract has three parts…

```
# funds-available:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes number of months, multiplies it by 1500, and subtracts the product from 50000
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  funds-available ( 1 ) is 50000 - (1 * 1500)
  funds-available ( 12 ) is 50000 - (12 * 1500)
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun funds-available( length, width ):
  50000 - (months * 1500)
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

The Design Recipe (Geometry - Rectangles)

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*

```
# Consumes 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)

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*

```
# Consumes length and width, and doubles of their sum to produce the perimeter
```
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*

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)

The Design Recipe (Geometry - Rectangular Prisms)

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*

```
# Consumes length, width, and height, and multiplies them to produce volume
```
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*

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)*

Directions: Write a function `rect-prism-sa` that takes in the width, length and height of a rectangular prism and calculates its surface area (the sum of the areas of each of its six sides)

Contract and Purpose Statement

Every contract has three parts…

```
# rect-prism-sa:: Number, Number, Number -> Number
```
function name / *domain* / *range*

```
# Consumes L, W, and H, calculates 2*L*W, 2*L*H, and 2*W*H
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  rect-prism-sa ( 10, 20, 30 ) is
    2 * (10 * 20 + (2 * (10 * 30))) + (2 * (20 * 30))
  rect-prism-sa ( 5, 12, 13 ) is
    2 * (5 * 12 + (2 * (5 * 13))) + (2 * (12 * 13))
end
```
function name / *input(s)* / *what the function produces*

Definition

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

```
fun rect-prism-sa( L, W, H ):
  2 * (L * W + (2 * (L * H))) + (2 * (W * H))
end
```
function name / *variable(s)* / *what the function does with those variable(s)*

The Design Recipe (Geometry - Circles)

Directions: Write a function `circle-area-dec` that takes in a radius and uses the decimal approximation of pi (3.14) to return the area of the circle.

Contract and Purpose Statement

Every contract has three parts…

`circle-area-dec`:: `Number` -> `Number`
function name *domain* *range*

Consumes the radius, squares it, multiplies it by 3.14 and returns the area
what does the function do?

Examples

Write some examples, then circle and label what changes…

`examples:`

`circle-area-dec ( 1 ) is 3.14 * num-sqr(1)`
function name *input(s)* *what the function produces*

`circle-area-dec ( 3 ) is 3.14 * num-sqr(3)`
function name *input(s)* *what the function produces*

`end`

Definition

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

`fun` `circle-area-dec( radius ):`
function name *variable(s)*

`3.14 * num-sqr(radius)`
what the function does with those variable(s)

`end`

Directions: Write a function `circumference` that takes in a radius and uses the decimal approximation of pi (3.14) to return the circumference of the circle.

Contract and Purpose Statement

Every contract has three parts…

`circumference`:: `Number` -> `Number`
function name *domain* *range*

Consumes the radius, doubles it, multiplies it by 3.14 and returns the circumference.
what does the function do?

Examples

Write some examples, then circle and label what changes…

`examples:`

`circumference ( 1 ) is (1 * 2) * 3.14`
function name *input(s)* *what the function produces*

`circumference ( 3 ) is (3 * 2) * 3.14`
function name *input(s)* *what the function produces*

`end`

Definition

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

`fun` `circumference( radius ):`
function name *variable(s)*

`(radius * 2) * 3.14`
what the function does with those variable(s)

`end`

The Design Recipe (Geometry - Cylinders)

Directions: Write a function `circle-area` that takes in a radius and uses the fraction approximation of pi ($\frac{22}{7}$) to return the area of the circle.

Contract and Purpose Statement

Every contract has three parts…

```
# circle-area:: Number -> Number
```
function name *domain* *range*

```
# Consumes the radius, squares it, multiplies it by 22/7 and returns the area
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  circle-area ( 1 ) is (22 / 7) * num-sqr(1)
  circle-area ( 3 ) is (22 / 7) * 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 ):
  (22 / 7) * num-sqr(radius)
end
```
function name *variable(s)*

what the function does with those variable(s)

Directions: Write a function `cylinder` that takes in a cylinder's radius and height and calculates its volume, making use of the function *circle-area*.

Contract and Purpose Statement

Every contract has three parts…

```
# cylinder:: Number, Number -> Number
```
function name *domain* *range*

```
# Consumes radius and height, and returns the product of the height and the area of the circle calculated
  using the radius and the function circle-area.
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  cylinder ( 10, 20 ) is 20 * circle-area(10)
  cylinder ( 200, 350 ) is 350 * circle-area(200)
end
```
function name *input(s)* *what the function produces*

Definition

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

```
fun cylinder(radius, height):
  height * circle-area(radius)
end
```
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- and y-coordinate and produces the next x-coordinate, which is 50 pixels to the left.

Contract and Purpose Statement

Every contract has three parts…

```
# update-danger:: Number, Number -> Number
```
function name / *domain* / *range*

```
# Consumes an x- and y-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, 100 ) is 160 - 50
  update-danger ( -85, 92 ) 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, y ):
  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 target's x- and y-coordinate and produces the next x-coordinate, which is 50 pixels to the right.

Contract and Purpose Statement

Every contract has three parts…

```
# update-target:: Number -> Number
```
function name / *domain* / *range*

```
# Consumes an x- and y-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, 286 ) is 130 + 50
  update-target ( -25, 110 ) 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!

The Design Recipe: Revenue & Cost

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)*

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

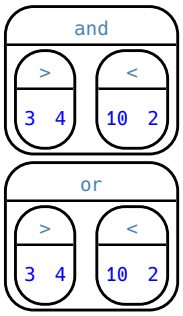
Write the definition, giving variable names to all your input values…

```
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 data type 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 greater 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 greater 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:



Boolean Functions

Explore the functions in the *Booleans Starter File* . What characteristics define them as Booleans?

Fill in the blanks below so that each of the five functions returns `true`

These are sample values, but answers will vary

1) `is-odd(5)`

2) `is-even(4)`

3) `is-less-than-one(0)`

4) `is-continent("Africa")`

5) `is-primary-color("red")`

Fill in the blanks below so that each of the five functions returns `false`

6) `is-odd(4)`

7) `is-even(3)`

8) `is-less-than-one(4)`

9) `is-continent("apple")`

10) `is-primary-color("orange")`

Simple Inequalities

Each inequality expression in the first column contains a number.
Decide whether or not that number is a solution to the expression and place it in the appropriate column.
Then identify 4 *solution* and 4 *non-solution* values for `x`.

- **Solutions** will make the expression `true`.
- **Non-Solutions** will make the expression `false`.

Challenge yourself to use negatives, positives, fractions, decimals, etc. for your `x` values.

Expression	4 solutions that evaluate to `true`	4 non-solutions that evaluate to `false`
`x > 2`	2.1, pi, 5, 1000	2, 0, -1, -333
`x <= -2`	-2, -3, -100, -1111	-1, 0, 3/4, 1200
`x < 3.5`	3, 2, 0, -1.5	3.5, 4, 10, 111
`x >= -1`	-1, 0, 5/3, 36.8	-2, -5, -10, -25
`x > -4`	-3.5, -1, 0, 17	-4, -5, -8.5, -1000
`x <> 2`	-2, 0, 1.9, 2.1	2 is the only non-solution

1) For which inequalities was the number from the expression part of the solution?

(<= x -2) and (>= x -1)

2) For which inequalities was the number from the expression not part of the solution?

(> x 2) and (< x 3.5) and (> x -4) and (<> x 2)

3) For which inequalities were the solutions on the left end of the number line?

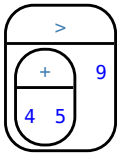
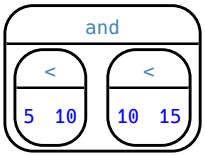
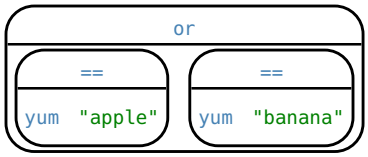
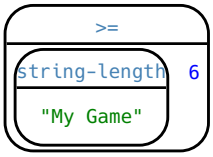
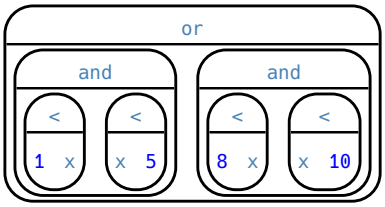
(<= x -2) and (< x 3.5) ... and (<> x 2) went both ways)

4) For which inequalities were the solutions on the right end of the number line?

(> x 2) and (>= x -1) and (> x -4) ... and (<> x 2) was at both ends

Converting Circles of Evaluation to Code

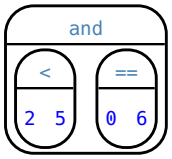
For each Circle of Evaluation on the left-hand side, write the code for the Circle on the right-hand side

	Circle of Evaluation	Code
1		`(4 + 5) > 9`
2		`(5 < 10) and (10 < 15)`
3		`(yum == "apple") or (yum == "banana")`
4		`string-length("My Game") >= 6`
5		`((1 < x) and (x < 5)) or ((8 < x) and (x < 10))`

Compound Inequalities — Practice

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

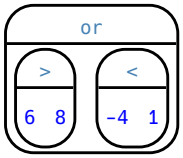
1) 2 is less than 5, and 0 is equal to 6




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

What will this evaluate to? *false*

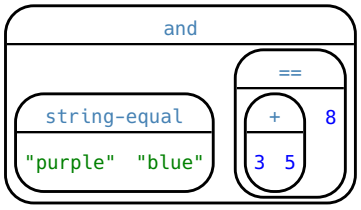
2) 6 is greater than 8, or -4 is less than 1




```
(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
```

Compound Inequalities: Solutions & Non-Solutions

For each Compound Inequality listed below, identify 4 *solutions* and 4 *non-solutions*. *If there are* ***no solutions*** *or the solution set includes* ***all real numbers*** *you can write that instead of making a list.*

- Solutions for ***intersections***, which use **and** will make both of the expressions `true`.
- Solutions for ***unions***, which use **or** will make at least one of the expressions `true`.

Pay special attention to the numbers in the sample expression! Challenge yourself to use negatives, positives, fractions, decimals, etc. for your `x` values.

The first two have been done for you - Answers will vary!

Expression	4 solutions that evaluate to `true`	4 non-solutions that evaluate to `false`
`x > 5` and `x < 15`	**6, 9.5, 12, 14.9**	**-2, 5, 15, 16.1**
`x > 5` or `x < 15`	**All real numbers**	**No non-solutions**
`x <= -2` and `x > 7`	*No solution*	*All real numbers*
`x <= -2` or `x > 7`	-2, -3.6, 8, 1000	7, -1, 0, 5
`x < 3.5` and `x > -4`	-3.5, 0, 1, 3	3.5, -4, -8, 400
`x < 3.5` or `x > -4`	*All real numbers*	*No non-solutions*
`x >= -1` and `x > -5`	-1, 0, 23, 400	-3, -5, -10, -542
`x >= -1` or `x > -5`	-4.5, -3, -1, 100	-5, -6, -28, -1000
`x < -4` and `x > 2`	*No solution*	*All real numbers*

1) Could there ever be a union with *no solutions*? Explain your thinking.

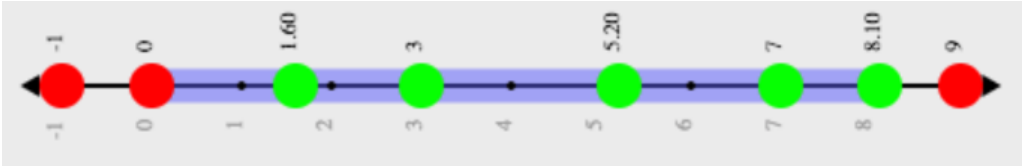
No. The solution set for each inequality includes a subset of the real numbers and the union of two inequalities will include more than the numbers in either of those subsets.

2) Could there ever be an intersection whose solution is *all real numbers*? Explain your thinking.

No. The solution set for each inequality includes a subset of the real numbers, so the list of numbers where the subsets overlap couldn't possibly include all of the real numbers.

Compound Inequality Functions

Each of the plots below was generated using the code `inequality(comp-ineq, [list: -1, 0, 1.6, 3, 5.2, 7, 8.1, 9] )`. With the exception of the example, each plot below was defined using the numbers 3 and 7. Write the code for how `comp-ineq` was defined for each plot in the space provided.



code: `fun comp-ineq(x): (x > 0) and (x <= 8.1) end`

code: `fun comp-ineq(x): (x > 3) and (x < 7) end`

code: `fun comp-ineq(x): (x > 3) or (x < 7) end`

code: `fun comp-ineq(x): (x < 3) or (x >= 7) end`

code: `fun comp-ineq(x): (x < 3) and (x > 7) end`

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 do you notice about the program?

2) What do you wonder?

3) What do you see when Sam is at (0,0)? Why is that? *Sam is at the center of the screen, because the origin is at the middle*

4) What changes as the butterfly moves left and right? *Sam's x-coordinate*

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

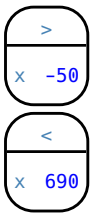
How far to the left and right can Sam go and still remain visible? Sam can go to the left until their x-coordinate is less than -40, and to the right until x > 680

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

5) Sam hasn't gone off the left edge of the screen as long as… `x > -50`

6) Sam hasn't gone off the right edge of the screen as long as… `x < 690`

7) 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)*

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
```
function name *input(s)*

```
is-safe-left(100) and is-safe-right(100)
```
what the function produces

```
is-onscreen ( -180 ) is
```
function name *input(s)*

```
is-safe-left(-180) and is-safe-right(-180)
```
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'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
fun cost(item):
  ask:
    | item == "hamburger" then: 6.0
    | item == "onion rings" then: 3.5
    | item == "fried tofu" then: 5.25
    | item == "pie" then: 2.25
    | otherwise: "Sorry, that's not on the menu!"
  end
end
```

1 **I notice…** *(sample responses) a function called* `cost` *, brackets, a function called* `string=?` *, numbers that look like prices, a contract and purpose statement, pizza toppings*	2 **I wonder…** *(sample responses) What is* `cond` *? Is this all that's on the menu? Can I add different toppings? How does the 'cost' function work? What are the brackets for? What does* `string=?` *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** *cond, string=?, [], else*

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? Number

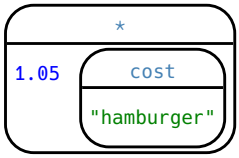
6) What is the name of its variable? menu-item

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

8) Try adding 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…

```
# order:: String -> Number
```
function name / *domain* / *range*

```
# Given an 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
  order ( "onion rings" ) is 3.5
  order ( "fried tofu" ) is 5.25
  order ( "pie" ) is 2.25
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:
  | item == "hamburger" then: 6
  | item == "onion rings" then: 3.5
  | item == "fried tofu" then: 5.25
  | item == "pie" then: 2.25
  | otherwise: "Sorry, not on the menu!"
  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 x- and 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, Number, String -> Number
```
function name — *domain* — *range*

```
# Consumes x, y and key. Produces new y depending on key pressed
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  update-player ( 170, 320, "up" ) is 320 + 20
  update-player ( 200, 100, "up" ) is 100 + 20
  update-player ( 60, 320, "down" ) is 320 - 20
  update-player ( 100, 495, "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( x, y, key ):
  ask:
  | "up" == key then: y + 20
  | "down" == key 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(128, "t") is        375
  update-player(128  "b") is        125
end
```

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

```
examples:
  update-player(480 "up") is        480
  update-player(0 "down") is         0
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(490 "up") is         0
  update-player(-10 "down") is       480
end
```

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

```
examples:
  update-player(128 "h") is      -1 * 128
  update-player(-391 "h") is     -1 * -391
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*

Definition

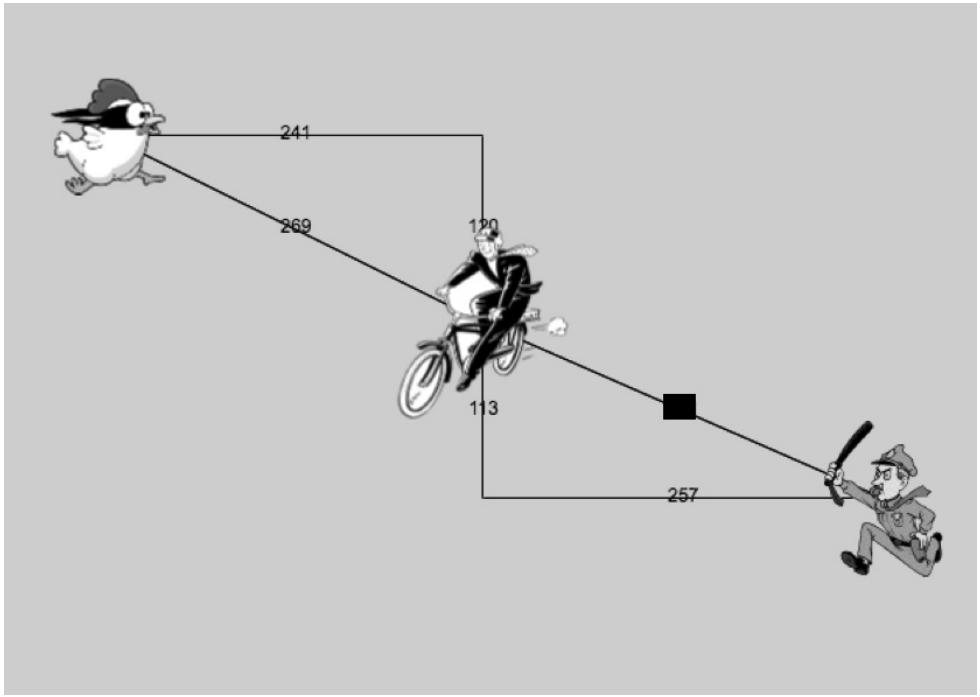
Write the definition, giving variable names to all your input values…

```
fun line-length( a, b ):
  ask:
  | a > b then: a - b
  | a < b then: b - a
  end
end
```
function name — *variable(s)*

Writing Code to Calculate Missing Lengths

In each of the game screenshots below, one of the distance labels has been hidden. Write the code to generate the missing distance on the line below each image. *Hint: Remember the Pythagorean Theorem!*


```
num-sqrt(num-sqr(174) + num-sqr(143))
```

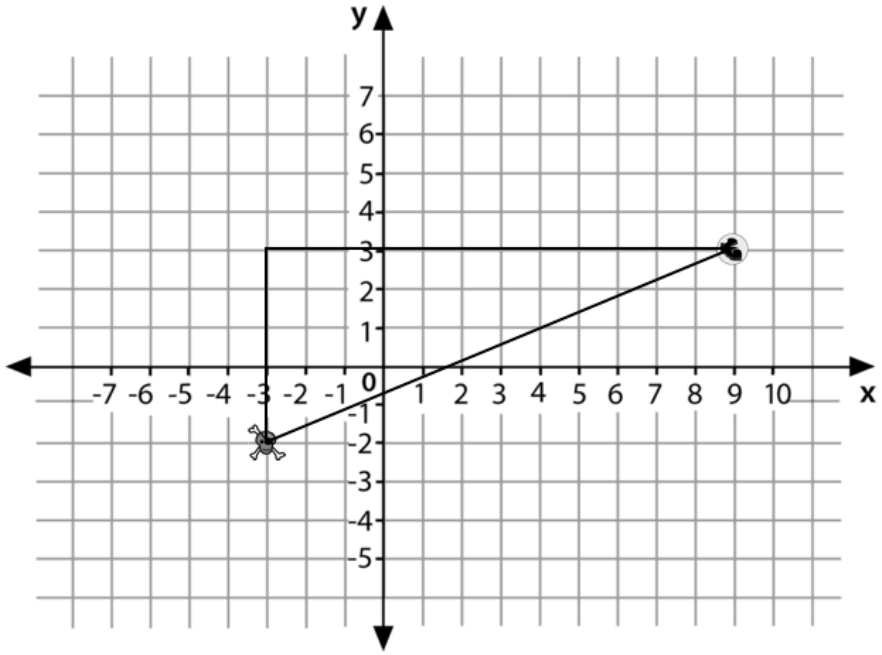



```
num-sqrt(num-sqr(113) + num-sqr(257))
```

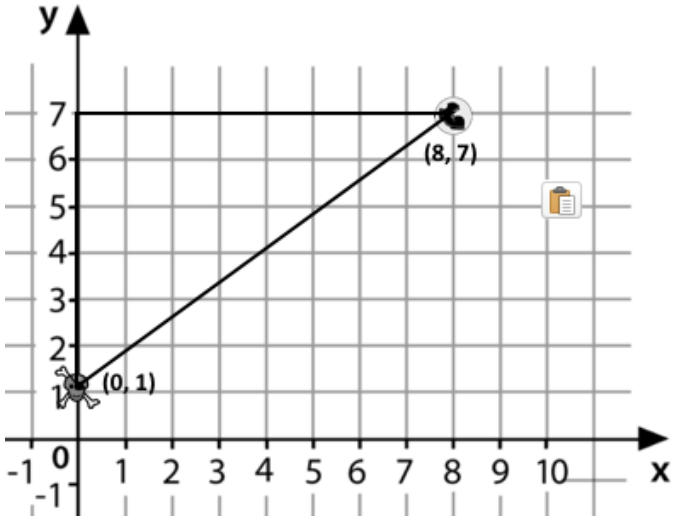
Distance on the Coordinate Plane

Distance between the pyret and the boot:

```
num-sqrt(num-sqr(line-length(9, -3)) + num-sqr(line-length(3, -2)))
```



Explain how the code works.



Now write the code to find the distance between this boot and pyret.

```
num-sqrt(num-sqr(line-length(8, 0)) + num-sqr(line-length(7, 1)))
```

or

```
num-sqrt(num-sqr(line-length(0, 8)) + num-sqr(line-length(1, 7)))
```

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

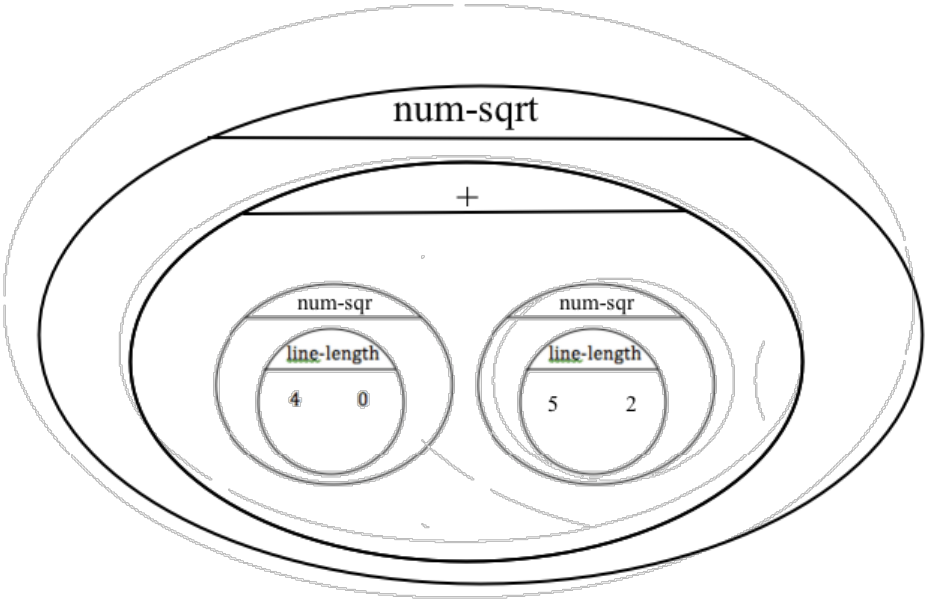
The distance between x_1 and x_2 is computed by `line-length(x1, x2)`. The distance between y_1 and y_2 is computed by `line-length(y1, y2)`. Below is the equation to compute the hypotenuse of a right triangle with those amount for legs:

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

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 y_2. The equation to compute the distance between these points is:

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

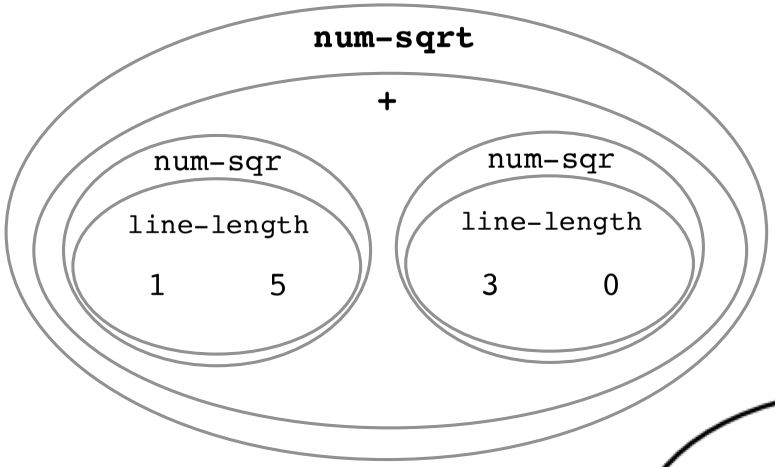
1. Translate the expression above, for (0,2) and (4,5) into a Circle of Evaluation below .



2. Convert the Circle of Evaluation to Code below .

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

Circle of Evaluation

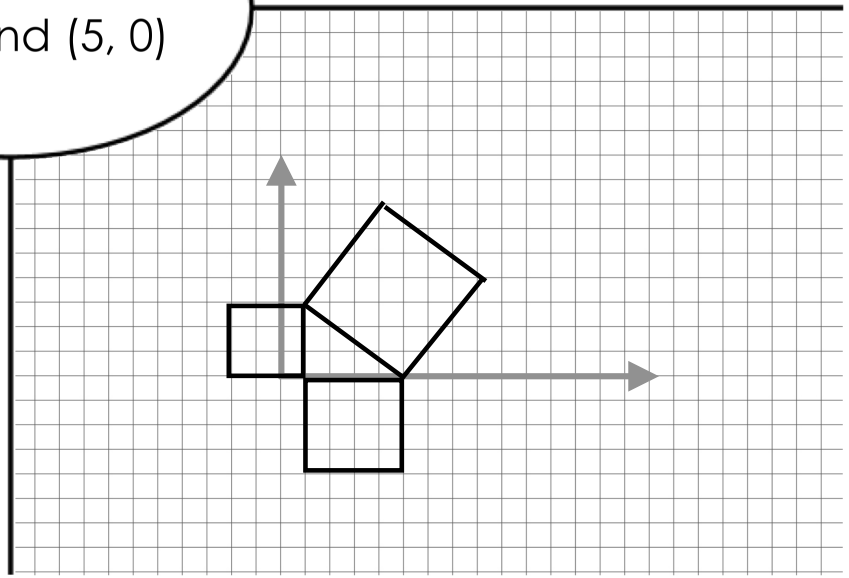


Code

```
(num-sqrt (+ (num-sqr (line-length 1 5))
             (num-sqr (line-length 3 0))))
```

Distance between (1,3) and (5, 0)

$$\text{sqrt}((1 - 5)^2 + (3 - 0)^2)$$
$$= \text{sqrt}((-4)^2 + 3^2)$$
$$= \text{sqrt}(16 + 9)$$
$$= \text{sqrt}(25)$$
$$= 5$$

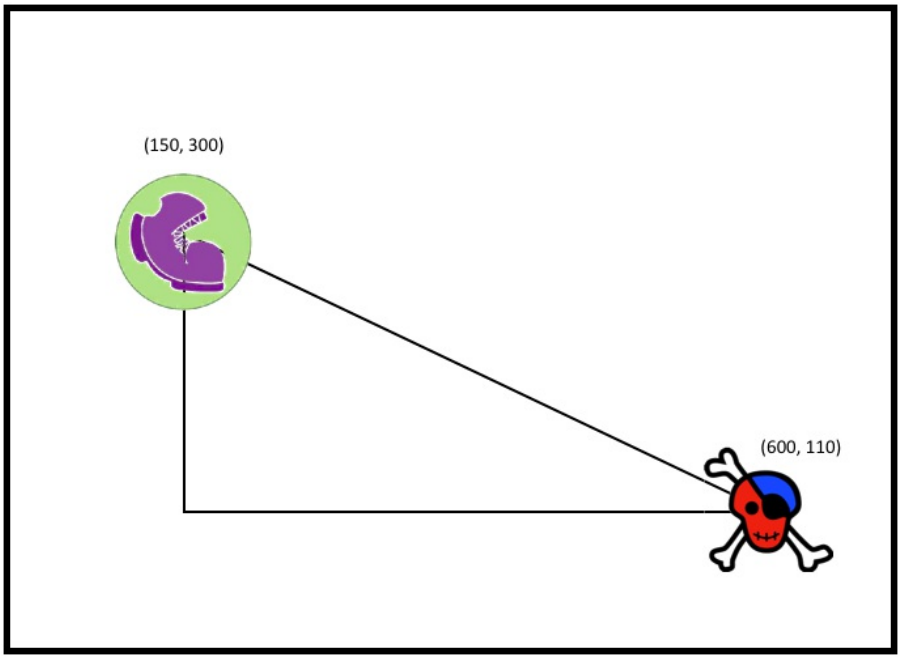


Computed distance between (1, 3) and (5, 0)

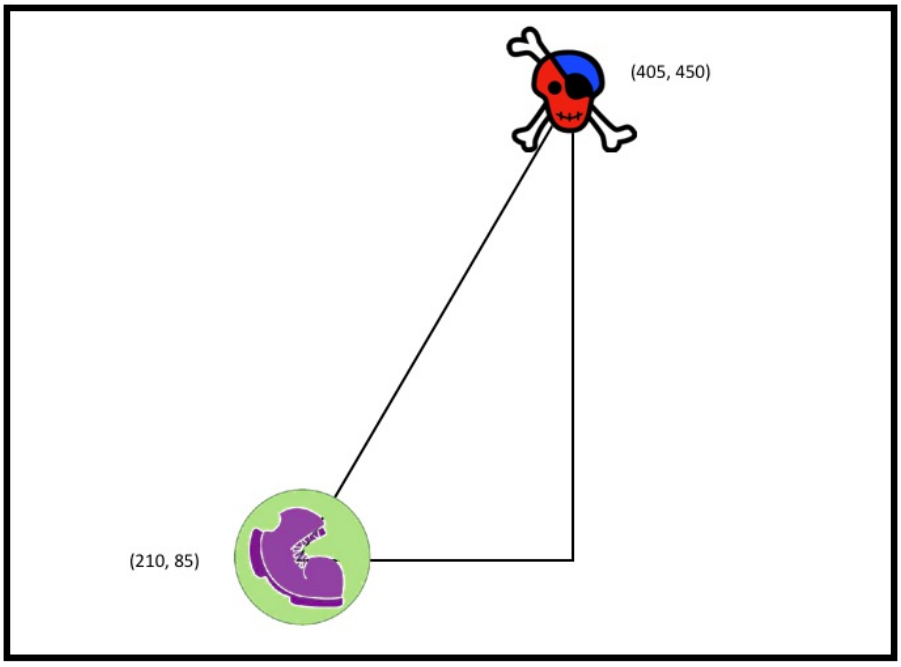
Graph

Distance From Game Coordinates

For each of the game screenshots, write the code to calculate the distance between the indicated characters. *The first one has been done for you.*

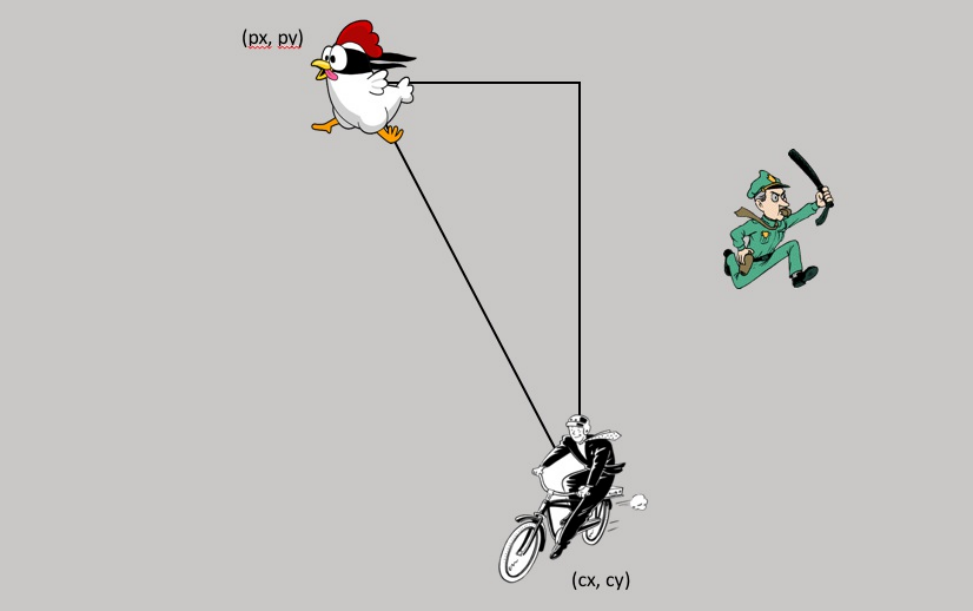



```
num-sqrt(num-sqr(line-length(600, 150)) + num-sqr(line-length(110, 300)))
```




```
num-sqrt(num-sqr(line-length(405, 210)) + num-sqr(line-length(450, 85)))
```

Distance (px, py) to (cx, cy)



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), 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*

```
# Consumes 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*

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

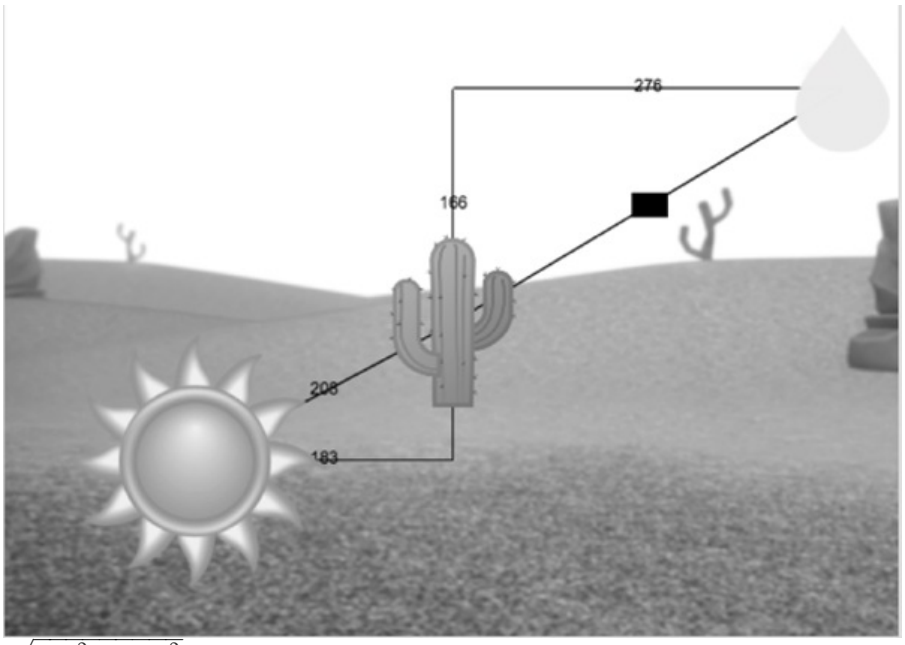
Definition

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)*

Comparing Code: Finding Missing Distances

For each of the game screenshots below, the math and the code for computing the covered distance is shown. Notice what is similar and what is different about how the top and bottom distances are calculated. Think about why those similarities and differences exist and record your thinking.



$\sqrt{166^2 + 276^2}$

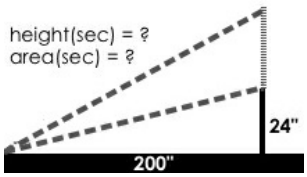
```
num-sqrt(num-sqr(166) + num-sqr(276))
```

$\sqrt{276^2 - 194^2}$

```
num-sqrt(num-sqr(276) - num-sqr(194))
```

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*

```
# Consumes two pairs of x/y coordinates and asks if the distance between them is less than 50
```
what does the function do?

Examples

Write some examples, then circle and label what changes…

```
examples:
  is-collide ( 0, 0, 30, 40 ) is distance(0, 0, 30, 40) < 50
  is-collide (25, 50, 250, 480) is
    distance(25, 50, 250, 480) < 50
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 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)`				
`# string-length`	`::`	`String`	`->`	`Number`
`string-length("Rainbow"}`				
`# string-contains`	`::`	`String, String`	`->`	`Boolean`
`string-contains("catnap", "cat")`				
`# triangle`	`::`	`Number, String, String`	`->`	`Image`
`triangle(80, "solid", "darkgreen")`				
`# star`	`::`	`Number, String, String`	`->`	`Image`
`star(50, "solid", "teal")`				
`# circle`	`::`	`Number, String, String`	`->`	`Image`
`circle(30, "outline", "fuchsia")`				
`# square`	`::`	`Number, String, String`	`->`	`Image`
`square(10, "outline", "red")`				
`# rectangle`	`::`	`Number, Number, String, String`	`->`	`Image`
`rectangle(20, 80, "solid", "gold")`				

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
`# rhombus`	`::`	`Number, Number, String, String`	`->`	`Image`
`rhombus(20, 50, "solid", "turquoise")`				
`# ellipse`	`::`	`Number, Number, String, String`	`->`	`Image`
`ellipse(30, 70, "outline", "lightblue")`				
`# text`	`::`	`String, Number, String`	`->`	`Image`
`text("I'm thankful for…", 50, "green")`				
`# regular-polygon`	`::`	`Number, Number, String, String`	`->`	`Image`
`regular-polygon(8, 40, "solid", "red")`				
`# right-triangle`	`::`	`Number, Number, String, String`	`->`	`Image`
`right-triangle(20, 50, "outline", "black")`				
`# isosceles-triangle`	`::`	`Number, Number, String, String`	`->`	`Image`
`isosceles-triangle(100, 30, "solid", "black")`				
`# radial-star`	`::`	`Number, Number, String, String`	`->`	`Image`
`radial-star(17, 50, 10, "solid", "orange")`				
`; star-polygon`	`:`	`Number, Number, Number, String, String`	`->`	`Image`
`star-polygon(100, 10, 3, "outline", "yellow")`				
`; triangle-sas`	`:`	`Number, Number, Number, String, String`	`->`	`Image`
`(triangle-sas 200 50 100 "solid" "blue")`				

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
# triangle-asa	::	Number, Number, String, String	->	Image
triangle-asa(100, 50, 200, "solid", "turquoise")				
# 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"))				

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
# beside	::	Image, Image	->	Image
beside(star(50, "solid", "orange"),circle(50, "solid", "green"))				
# or	:	Boolean, Boolean	->	Boolean
(<= 3 2) or (>= 3 2)				
# and	:	Boolean Boolean	->	Boolean
(<= 3 2) and (>= 3 2)				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				
#	::		->	
#				