BOOTSTRAP
www.bootstrapworld.org

Match the Shape Activity

Log in to https://www.wescheme.org/ and open https://www.wescheme.org/openEditor?publicId=KozEydZjs2, and you should see the code. You'll want to then click Remix to save a copy of the program to your account. (Don't see Remix? You might not be logged in.)

Click Run above the code. You'll see an orange triangle on a coordinate plane. Take a look at the code and you'll see that the only actual expression being evaluated is `(render shapeB)`. Everything else is a comment.

1. What do you notice about `shapeB`? Add a comment to the code that describes `shapeB` as thoroughly as possible. Here's what it might look like:

```
; shapeB : an orange obtuse triangle with vertices at (4,2), (2,4),
and (-2,1)
```

1a. But what *is* `shapeB`? Type `shapeB` into the interactions area. You'll see some new code that starts with (shape …) You don't have to know how this code works, only that `shapeB` is an instance of a new datatype called a **Shape.** However, this isn't a built-in datatype like Numbers, Strings, or Images. We've defined this datatype for you to use inside this file. (For more information on creating your own custom datatypes, check out Bootstrap:Reactive!)

A Shape is a piece of data consisting of a list of positions, or posns for short. You don't need to know how posns or lists work, only that `shapeA, shapeB, center-square`, and the other shapes indicated in this file are instances of the datatype **Shape**.

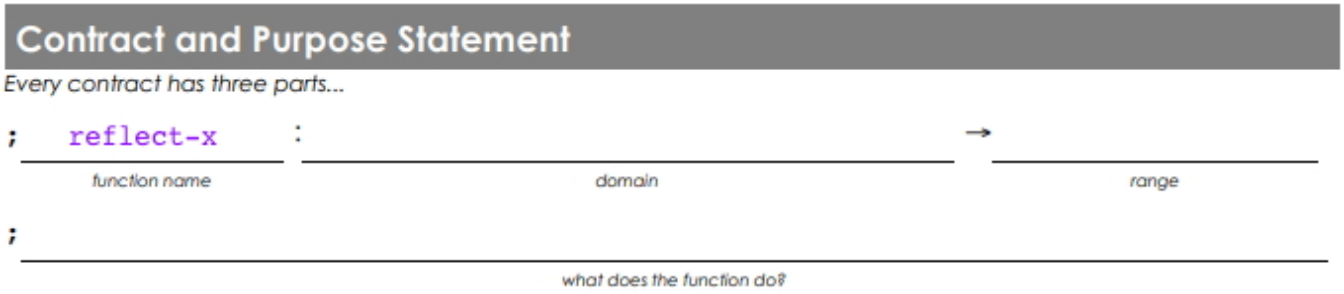
Evaluating the name of a shape and seeing only the code behind it makes it hard to visualize what the shape actually *looks* like. Thankfully we have the `render` function, which consumes a Shape, and produces an *image* of that shape on a coordinate grid.

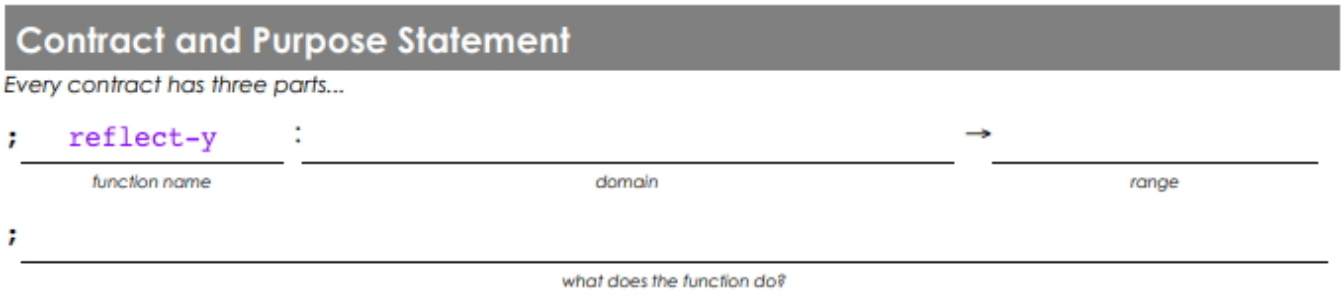
```
; render : Shape -> Image
; Add the given Shape to the background grid at the correct
coordinates
```

2: Type `(render shapeA)` in the interactions area. Render the rest of the shapes listed in the Sample Shapes section and add comments to describe them.

3: Type the following command: `(render (reflect-x shapeA))`. Then try `(render (reflect-y shapeA))`. What do you notice about what happened in each case? What do you wonder?

Try to write a contract and a purpose statement for these two new functions, `reflect-x` and `reflect-y`.



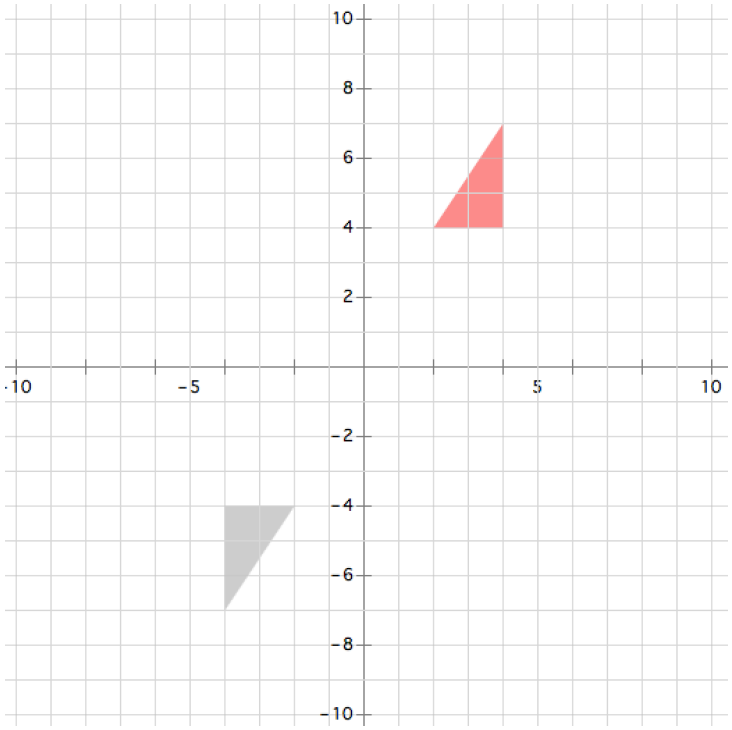


Add the contracts and purpose statements for the two reflect functions to your program.

4a: Draw the circle of evaluation for an expression that will reflect a shape over both the x- and y-axes. How did you figure out what to do?

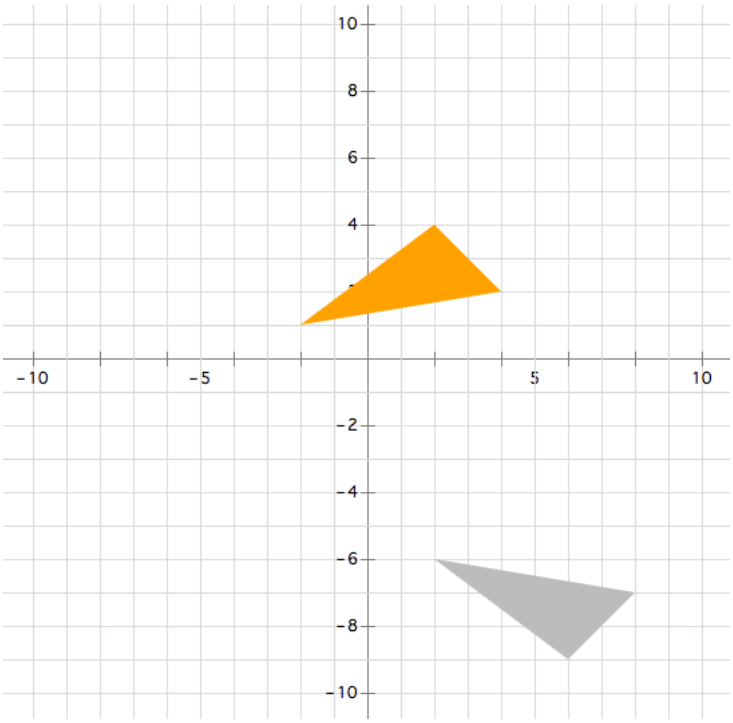
4b: Convert your circle of evaluation to a Racket expression and try it!

5: You've seen the `render` function, which draws a coordinate grid and the appropriate shape in the Interactions area. There is also a `render-both` function. Try the command `(render-both shapeB (reflect-x shapeB))` and see what the function does. (Add a contract and purpose statement for it to your program.) Then, use the `render-both` function to duplicate the picture below.



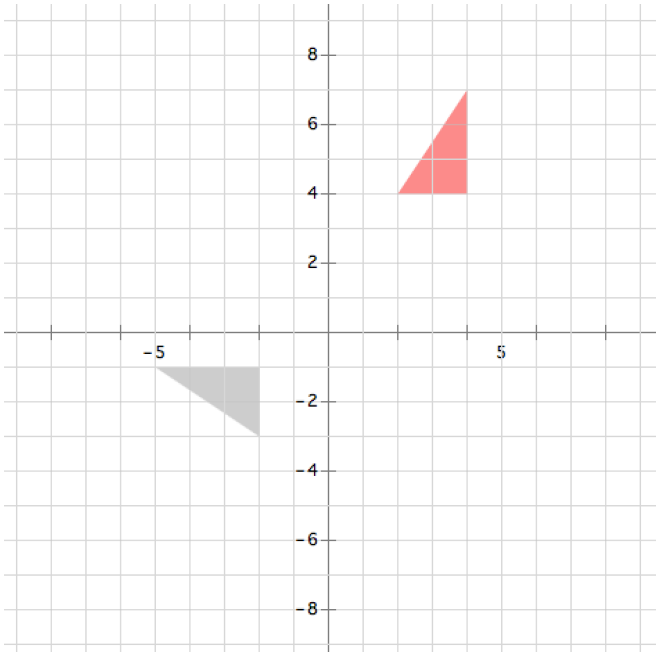
6: Let's explore the `translate` function. Type this in the Interactions area and see what you can figure out about how this function works:
`(render (translate shapeA 4 -5))`. Once you've figured it out, add a contract and purpose statement to your program.

7: Use the `reflect` and `translate` functions to duplicate this result:



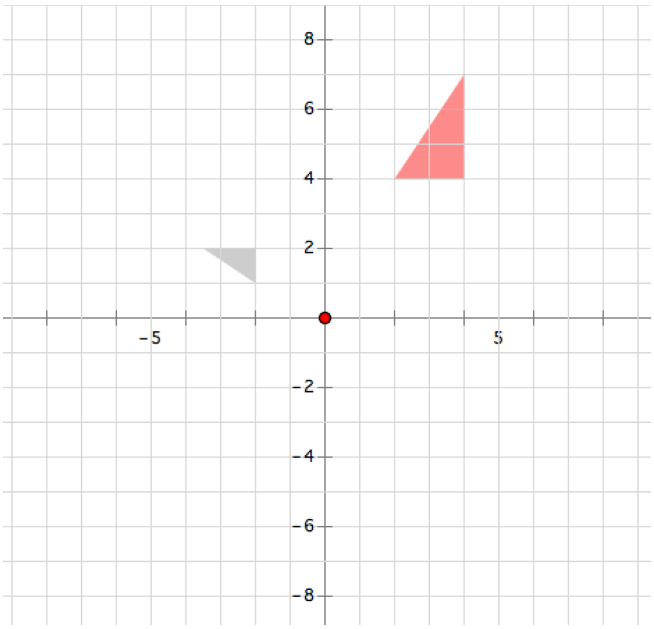
8: Explore the `rotate-shape` function, starting with this example:
`(render (rotate-shape shapeB 6 0 90))`

9. Try to duplicate this result:



10. Explore the `dilate` function, starting with this example:
`(render (dilate shapeB .5))`

11. Try to duplicate this result:



12. Create a result for your classmates to try to duplicate using no more than three transformations. Take a screenshot of the result you get in the Interactions area to share with your classmates.