Function Composition

(Also available in [WeScheme](#))

Students learn to combine image transformation functions as well as to describe the order of operations involved in algebraic function compositions such as f(g(h(x))) using Circles of Evaluation.

Lesson Goals	Students will be able to: • Use functions as building-blocks, composing them to achieve a desired affect • Diagram function composition using the Circles of Evaluation • Compose functions when programming
Student-facing Goals	• Let's learn to use functions as building blocks that can be combined or composed to achieve desired outcomes. • Let's use Circles of Evaluation to show how functions can be composed.
Prerequisites	• [Simple Data Types](#) • [Order of Operations](#) • [Contracts](#)
Materials	• [PDF of all Handouts and Page](#) • [Lesson Slides](#) • [Printable Lesson Plan](#) (a PDF of this web page)
Supplemental Materials	• [Additional Printable Pages for Scaffolding and Practice](#) • *[Function Cards (Desmos)](#)* • *[Function Composition Matching Activity (Desmos)](#)* • [Random Integer Generator](#) • [Project: Create Your Own Logo](#)

Key Points For The Facilitator	• Check frequently for understanding of *data types* and *contracts* during this lesson and throughout subsequent lessons. • When students encounter errors, encourage them to check their contracts and show their work using Circles of Evaluation.

Glossary

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

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

image :: a type of data for pictures

Composing Functions   *10 minutes*

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

Launch

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

Have students practice reading contracts by asking them questions about one or two of these cards. For example: "pick a card at random: according to its Contract, what is the name? Domain? Range?"

These cards represent a collection of functions, each of which takes an input and produces an output. I can start with the number `4` , for example, and give it to the function `add6` . What will the output be? *10* **I can also *compose* functions, meaning that the output of one is immediately passed into another. For example, I could compose `add6` and `double` , so the `10` gets passed into the next function, and doubled to produce `20` . What would happen if I composed `add6` with `double` *and* with `half` ?** *10*

> - For each of the starting numbers on the board, your job is to figure out which functions to compose in order to get to the end.
> - You will need to use some functions more than once, and that's okay!

Investigate

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

Other ways to play…

- What solution uses the smallest number of **steps**?
- What solution uses the smallest number of **cards**?
- Show students a list of starting and ending numbers… *then ask them to give a card back.*
- Show students a list of starting and ending numbers… *then ask them which are the most valuable*
- Show students a list of starting and ending numbers… *then give them a blank card and let them come up with a new function*

Synthesize

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

Diagramming Function Composition   *15 minutes*

Overview

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

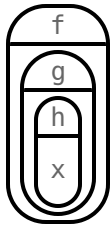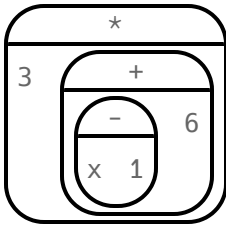## Launch

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

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

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

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

Function Composition	Order of Operations

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

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

Investigate

> Turn to [Diagramming Function Composition](#) to practice writing, translating and evaluating Circles of Evaluation of composed functions.

More practice is available on <u>Function Composition: Matching</u> and <u>Diagramming Function Composition (2)</u>.

Synthesize

- Do $f(g(h(x)))$ and $g(h(f(x)))$ evaluate to the same thing? Why or why not?
  - *No, they do not. Order matters!*

Composing Functions in Code   *20 minutes*

Overview

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

Launch

Explain to students that the functions available to them in Pyret can be composed just like the Function Cards from the activity. Their job as programmers is to figure out how to compose functions to achieve their goals in the most clever or elegant way possible.

Investigate

- Have students open [code.pyret.org (CPO)](code.pyret.org) in their browser, and "Sign In" using a valid Google account (Gmail, Google Classroom, YouTube, etc.) and their password for that account.
- This will take them to the "Programs" page. This page is empty - they don't have any programs yet!
- Have them open a new program by clicking "File" -> "New" and save it as "Function Composition"
- Complete [Function Composition — Green Star](), in which you will draw circles of evaluation to help you write expressions to compose a series of images.
- Be sure to use the **Definitions Area** (left side) for code you want to keep and the **Interactions Area** (right side) to test code or try out new ideas.

When students are finished, check their work, and ask them to change the color of all of the stars to "gold" or another color of their choosing.

Now, turn to [Function Composition — Your Name]() in which you will create a text *image* of your name and experiment with other functions.

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

Synthesize

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

- Because the `text` function produces an Image, which is then used as the input for `flip-horizontal`.

Fun with Images!

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

Our [Making Flags](#) also dives deeper into image composition.

Composing Multiple Ways *Optional*

Overview

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

Launch

As is often true with solving math problems, there is more than one way to get the same composed image.

- Suppose I wrote the code: `scale(3, circle(50, "solid", "red"))`.
- What's another line of code I could write that would produce the exact same image?
  - `circle(150, "solid", "red")`

Investigate

Complete [More than one way to Compose an Image!](#)

When students have completed the worksheet, explain that there is a special function that lets us test whether or not two images are equal:

`images-equal :: Image, Image -> Boolean`

Invite students to use the above function to test whether all of the expressions that they wrote successfully build the same images.

Synthesize

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