Composing Table Operations

(Also available in CODAP)

Students learn how to compose functions that operate on tables.

Lesson Goals	Students will be able to… • Compose table operations to create more sophisticated analyses. • Diagram their composition to make sense of the order of operations • Find bugs when table operations are not composed in the correct order
Student-facing Lesson Goals	• Let's practice combining table functions
Prerequisites	• Simple Data Types • Contracts • Functions: Contracts, Examples & Definitions • Contracts: Making Tables and Displays • Solving Word Problems with the Design Recipe • Filtering and Building
Materials	• PDF of all Handouts and Page • Animals Starter File • Lesson Slides • Printable Lesson Plan (a PDF of this web page)
Supplemental Materials	• Additional Printable Pages for Scaffolding and Practice

Glossary

circle of evaluation :: a diagram of the structure of a mathematical expression

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

Design Recipe Practice *25 minutes*

Overview

Students apply the Design Recipe to make table functions that operate on rows of the Animals Dataset. These become the basis of the composition activity that follows.

Launch

When filtering rows or building columns, we need to write functions . *This should be done carefully!* We want our results to be rock solid and accurate, especially if they're going to be used in ways that affect the world around us.

The Design Recipe is a sequence of steps that helps us document, test out, and write functions that let us dig deeper into our data, and analyze it more carefully. It's important for this to be like second nature, so let's get some practice using it.

Investigate

- Open your saved Animals Starter File.
- Define the functions on The Design Recipe: is-dog / is-female and The Design Recipe: is-old / name-has-s.

Optional: Combining Booleans

Suppose we want to build a table of Animals that are fixed *and* old, or a table of animals that are cats *or* dogs?

By using the `and` and `or` operators, we can *combine* Boolean tests , as in: `(1 > 2) and ("a" == "b")` . This is handy for more complex programs! For example, we might want to ask if a character in a video game has run out of health points *and* if they have any more lives. We might want to know if someone's ZIP Code puts them in Texas or New Mexico. When you go out to eat at a restaurant, you might ask what items on the menu have meat and cheese.

For many of the situations where you might use `and` , there's actually a much more powerful mechanism you can use, called *Composition* !

Synthesize

- Did you find yourselves getting faster at using the Design Recipe?
- What patterns or shortcuts are you noticing, when you use the Design Recipe?

Composing *25 minutes*

Overview

Students learn to diagram expressions using the Circles of Evaluation. This tool has deep roots in both Order of Operations and Function Composition, and math teachers may want to take a detour through one or both of these lessons to support those learning goals.

Launch

We already know how to filter, sort, and build columns - but what if we want to do *multiple things, all at once?* Sorting, Filtering and Building are powerful operations, but when they are *combined* they become even more powerful!

A journalist comes to the shelter who wants to write a story about a successful pet adoption — but she has a very specific set of criteria. The reporter wants to report on the adoption of an animal that weighs **more than 9 kilograms** (they don't use "pounds" in Britain!).

- To provide her with this data, what operations do we need to do to our dataset?
  - *We need to filter, showing only rows that are greater than 9kg. We also need to add a column that shows weight in kilograms, dividing pounds by 2.2.*

Order matters: Build , Filter, Sort.

- What do you think will happen if we try to filter animals that weigh more than 9kg, before actually building a `"kilos"` column?
  - *(Sample responses:) It will crash! The computer won't like it!*

If we use our functions in the wrong order (trying to filter by a column that doesn't exist yet), we might wind up crashing the program. But even worse, the program might run but produce nonsensical results!

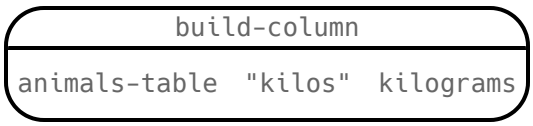
One way to organize our thoughts is to diagram what we want to do, using the ***Circles of Evaluation***. The rules are simple:

1) Every Circle must have one - and only one! - function , written at the top.

2) The arguments of the function are written left-to-right, in the middle of the Circle.

Values like Numbers, String, and Booleans are still written by themselves. It's only when we want to _use a function _ that we need to draw a Circle, and write the values inside from left-to-right.

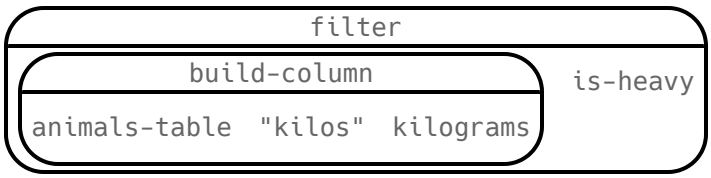
Let's try diagraming what we need to do for the journalist, using the Circles of Evaluation. We always build first, so let's start there. According to the ***Contract***, we know the name of the function is `build-column`, and it needs three arguments: the animals table, the name of the new column `"kilos"`, and the `kilograms` function.



But we also need to filter by that new column, so that we only have animals weighing more than 9kg! That means we need *another* Circle of Evaluation. We know `filter` goes at the top. But what table are we using for the first argument? It can't be the animals-table again, because that doesn't have a `"kilos"` column.

3) Circles can contain other Circles!

Our first Circle of Evaluation *produces a table*, and that's the one we want to use as the first input to `filter`!



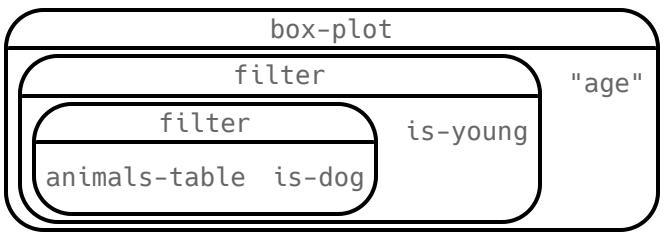
Investigate

- Complete Composing Table Operations.
- *Optional:* For more of a challenge, tackle Composing Table Operations: Order Matters!

Circles of Evaluation let us think and plan, without worrying about small details.

Sometimes, the hardest part of solving a problem is knowing what you want to do, rather than worrying about how to do it. For example, sometimes solving an equation is a lot easier than *setting it up in the first place*. Circles of Evaluation give us an opportiunity to think through what we want to do, before getting in front of the computer and worrying about how to do it.

Armed with these tools, we can do some pretty complex analysis! We can even think of data displays as another kind of table operation. What will this Circle of Evaluation produce?



To convert a Circle of Evaluation into code, **we start at the outside and work our way in**. After each function we write a pair of parentheses, and then convert each argument inside the Circle. The code for this Circle of Evaluation would be

`box-plot(filter(filter(animals-table, is-dog), is-young), "age")`.

- Type this into Pyret and see what you get!
- Draw the Circle of Evaluation showing how to make a bar chart showing the species in the shelter, *but only for old animals*. Then convert it to code and type it into Pyret.
- For practice converting Circles of Evaluation into code, complete From Circles to Code.

Teaching Tip

Use different color markers to draw the Circles of Evaluation, and then use those same colors when writing the code. This helps make the connection between Circles and code clearer.

Complete Planning Table Operations.

Synthesize

Review student answers to Planning Table Operations.

Was it helpful to think about the Circles, without worrying about Pyret? Why or why not?

Additional Exercises

From Circles to Code (2)