Composing Table Operations

(Also available in Pyret)

Students learn how to compose Transformers, which act as functions operating 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 Transformers
Prerequisites	• Introduction to Data Science • Exploring CODAP • Dot Plots and Bar Charts • Introduction to Transformers: Filter • More Transformers: Transform Attribute and Build Attribute
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

Design Recipe Practice *25 minutes*

Overview

In this lesson students build on what they have already learned to formalize their understanding of the Design Recipe, and then gain fluency with using the Design Recipe.

Launch

When filtering rows or building columns, we need to write Transformer expressions. *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 Transformer expressions 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.

Let's look at Design Recipe together.

First, we need to decide which Transformer to use: Filter, Transform, or Build. Based on the Transformer's already-provided name, students should deduce that they will use Filter. They can then record the Transformer's name on the line.

Transformer (check one)	☐ Filter	☐ Transform	☐ Build

filter-is-dog

Transformer name

Next, we provide **example tables**. In this case, we want to know the animals' names and their species, so we write down those column names. We want to list a few different animals - at least one that is a dog, and at least one that is not - to represent the variety of animals on the table. Then we think about what our transformed table will look like:

- Will Sasha be on the new table? *No, Sasha is a cat. We only want dogs!*
- Will Fritz be on the new table? *Yes, Fritz is a dog.*
- Will Toggle be on the new table? *Yes, Toggle is a dog.*

Example Tables

What gets filtered/transformed/built? In the sample tables below, add the relevant columns.

Original Table

name	species
sasha	cat
fritz	dog
toggle	dog

Transformed Table

name	species
fritz	dog
toggle	dog

Now, we are ready to drill down on the contents of our Transformer.

- First - the **contract**, which requires a domain (what type of data will we provide) and a range (what type of data will be produced). Whenever we are filtering, we can expect the Contract to be the same: Row -> Boolean.
- Next, we need a clear **purpose statement**, which describes what the expression does to each row. In this case, the expression will consume an animal and compute whether the species is "dog" - as our example tables (above) demonstrate!
- And finally, we enter our **expression**, in this case: `species = "dog"`.

Contents (Contract, Purpose Statement, and Expression)

Row	->	Boolean
Domain		*Range*

consumes an animal and computes whether the species is "dog"

Purpose: what does the formula do for each row?

```
species = "dog"
```

i.e. Weight < 20 or Species = 'rabbit'. Pay careful attention to capitalization and quotation marks.

Each time students encounter a new word problem, we encourage working through it with paper and pencil, as above; the Design Recipe slows down students' thinking and encourages them to reason through each scenario fully rather than guessing haphazardly.

Investigate

- Open your saved Animals Starter File.
- Define the Transformers and expressions on Design Recipe 2.

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 on a single Transformer, like `Filter`. Once we've opened the `Filter` Transformer, we would tell CODAP to keep all rows that satisfy `Species = "cat" and Species = "dog"`. 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.

When we want to compose *different* Transformers, however, this strategy will not work. We'll need to find another way!

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 / Transform, 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 Transformers 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! - Transformer type, written at the top.

2) The arguments of the Transformer 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 Transformer* 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. We know that our transformer needs three things: the animals table, the name of the new column `"kilos"` and the formula expression.

```
build-attribute
animals-table  "kilos"  pounds/2.205
```

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`!

```
filter
  build-attribute                  kilos>10
  animals-table  "kilos"  pounds/2.2
```

Investigate

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

A perk of composing Transformers is that everything is just a "view" of the original data, rather than a *change* made to that data. Changes can cause tables to go out of sync, resulting in hard-to-find bugs and invalid results. With Transformers, any updates made to the original dataset will flow through the composition, keeping everything in sync. Transformers can also be reused, eliminating duplicate work.

Tip: Saving Transformers and Renaming Tables

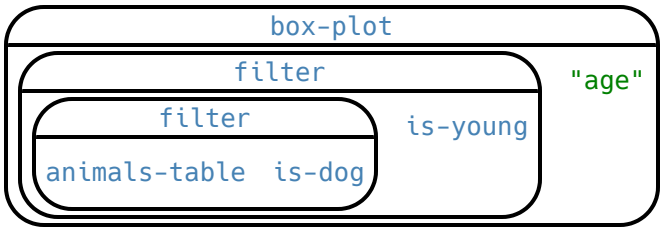
Saving a particular configuration of a Transformer is useful so that the Transformer can be easily accessed in the future. When we save a Transformer, we want to give it a useful name and purpose statement for ease of use later.

We also encourage students to rename tables descriptively. By the end of this exercise, the table students create will have quite a lengthy name: `(weight-in-kg(filter-if-light(Animals-Dataset)))`. That's a lot of parentheses! As an alternative, students might consider using renaming the table. For instance, `light-animals-in-kg` might be a more useful table name, here.

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?



Complete Planning Table Operations.

Synthesize

Review student answers to Planning Table Operations.

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

Additional Exercises

From Circles to Transformers