










Defining Table Functions

Students continue practicing the Design Recipe, writing helper functions to filter rows and build columns in the Animals Dataset, using Methods.

Prerequisites	Table Methods																				
Relevant Standards	Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).																				
Lesson Goals	Students will be able to... <ul style="list-style-type: none">• write custom helper functions to filter the animals table• write custom helper functions to build on the animals table																				
Student-facing Lesson Goals	<ul style="list-style-type: none">• Let’s practice writing functions to filter and expand our tables.																				
Materials	<ul style="list-style-type: none">• Lesson Slides (Google Slides)• Computer for each student (or pair), with access to the internet• Student workbook, and something to write with• All students should log into CPO and open the "Table Methods Starter File" they saved from the prior lesson. If they don’t have the file, they can open a new one																				
Preparation	<ul style="list-style-type: none">• Make sure all materials have been gathered• Decide how students will be grouped in pairs																				
Supplemental Resources																					
Language Table	<table><tr><th>Types</th><th>Functions</th><th>Values</th></tr><tr><td>Number</td><td>num-sqrt, num-sqr</td><td>4, -1.2, 2/3</td></tr><tr><td>String</td><td>string-repeat, string-contains</td><td>"hello", "91"</td></tr><tr><td>Boolean</td><td>==, <, <=, >=, string-equal</td><td>true, false</td></tr><tr><td>Image</td><td>triangle, circle, star, rectangle, ellipse, square, text, overlay, bar-chart, pie-chart, bar-chart-summarized, pie-chart-summarized</td><td>  </td></tr><tr><td>Table</td><td>count, .row-n, .order-by, .filter, .build-column</td><td></td></tr></table>			Types	Functions	Values	Number	num-sqrt, num-sqr	4, -1.2, 2/3	String	string-repeat, string-contains	"hello", "91"	Boolean	==, <, <=, >=, string-equal	true, false	Image	triangle, circle, star, rectangle, ellipse, square, text, overlay, bar-chart, pie-chart, bar-chart-summarized, pie-chart-summarized	  	Table	count, .row-n, .order-by, .filter, .build-column	
Types	Functions	Values																			
Number	num-sqrt, num-sqr	4, -1.2, 2/3																			
String	string-repeat, string-contains	"hello", "91"																			
Boolean	==, <, <=, >=, string-equal	true, false																			
Image	triangle, circle, star, rectangle, ellipse, square, text, overlay, bar-chart, pie-chart, bar-chart-summarized, pie-chart-summarized	  																			
Table	count, .row-n, .order-by, .filter, .build-column																				

Defining Lookup Functions

25 minutes

Overview

Students continue practicing the Design Recipe, by writing functions to answer **Lookup Questions**.

Launch

Take two minutes to find all the fixed animals by hand. Turn to [The Animals Dataset](#), and walk down the table one row at

a time, putting a check next to each animal that is fixed.

To do this activity, what kind of question were you asking of each animal? Was it a **Lookup**, **Compute**, or **Relate** question? You went through the table one row at a time, and for **each row** you did a lookup on the `fixed` column.

Have students type the code that will look up if `animalX` is fixed or not, then do the same with `animalY`. Suppose we wanted to do this for every animal in the table? This seems really repetitive, doesn't it? We would keep typing the same thing over and over, but all that's really changing is the animal. Wouldn't it be great if Pyret had a function called `lookup-fixed`, that would do this for us?

Fortunately, we already know how to define functions using the Design Recipe!

Turn to [The Design Recipe \(Page 28\)](#) in your Student Workbook.

Step 1: Contract and Purpose

The first thing we do is write a Contract for this function. You already know a lot about contracts: they tell us the Name, Domain and Range of the function. Our function is named `lookup-fixed`, and it consumes a row from the animals table. It looks up the value in the `fixed` column, which will always be a Boolean. A Purpose Statement is a description of what the function does:

```
# lookup-fixed :: (r :: Row) -> Boolean
# Consumes an animal, and lookup the value in the fixed column
```

Since the contract and purpose statement are notes for humans, we add the `#` symbol at the front of the line to turn it into a comment. Note that we used "lookup" in the purpose statement!

Be sure to check students' contracts and purpose statements before having them move on.

Step 2: Write Examples

Writing examples for Lookup questions is really simple: all we have to do is look up the correct value in the table, and then write the answer!

```
# lookup-fixed :: (r :: Row) -> Boolean
# Consumes an animal, and looks up the value in the fixed column
examples:
  lookup-fixed(animalX) is true
  lookup-fixed(animalY) is false
end
```

Step 3: Define the Function

When defining the function, we replace the answer with the lookup code.

```
# lookup-fixed :: (animal :: Row) -> Boolean
# Consumes an animal, and looks up the value in the fixed column
examples:
  lookup-fixed(animalX) is true
  lookup-fixed(animalY) is false
end
fun lookup-fixed(r): r["fixed"]
end
```

Investigate

For practice, try using the Design Recipe to define another lookup function.

- Use the Design Recipe to solve the word problem at the bottom of [The Design Recipe \(Page 28\)](#).
- Type in the Contract, Purpose Statement, Examples and Definition into the Definitions Area.

- Click “Run”, and make sure all your examples pass!
- Type `lookup-sex(animalX)` into the Interactions Area.

Defining Compute Functions

25 minutes

Overview

Students define functions that answer **Compute Questions**, again practicing the Design Recipe.

Launch

We’ve only been writing **Lookup Functions**: they consume a Row, look up one column from that row, and produce the result as-is. And as long as that row contains Boolean values, we can use that function with the `.filter` method.

But what if we want to filter by a Boolean expression? For example, what if we want to find out specifically whether or not an animal is a cat, or whether it’s young? Let’s walk through an example of a Compute Function using the Design Recipe, by turning to [The Design Recipe \(Page 29\)](#).

Suppose we want to define a function called `is-cat`, which consumes a row from the `animals-table` and returns true if the animal is a cat.

- Is this a Lookup, Compute or Relate question?
- What is the name of this function? What are its Domain and Range?
- Is Sasha a cat? *What did you do to get that answer?*

To find out if an animal is a cat, we look-up the species column and check to see if that value is *equal* to `"cat"`. Suppose `animalX` is a cat and `animalY` is a dog. What should our examples look like? **Remember: we replace any lookup with the actual value, and check to see if it is equal to `"cat"`.**

```
# is-cat :: (r :: Row) -> Boolean
# Consumes an animal, and compute whether the species is "cat"
examples:
  is-cat(animalX) is "cat" == "cat"
  is-cat(animalY) is "dog" == "cat"
end
```

Write two examples for your defined animals. Make sure one is a cat and one isn't!

As before, we'll use the pattern from our examples to come up with our definition.

```
# is-cat :: (r :: Row) -> Boolean
# Consumes an animal, and compute whether the species is "cat"
examples:
  is-cat(animalX) is "cat" == "cat"
  is-cat(animalY) is "dog" == "cat"
end
fun is-cat(r): r["species"] == "cat"
end
```

Don't forget to include the lookup code in the function definition! We only write the actual value for our examples!

Investigate

- Type this definition – and its examples! – into the Definitions Area, then click “Run” and try using it to filter the `animals-table`.
- For practice, try solving the word problem for `is-young` at the bottom of [The Design Recipe \(Page 29\)](#).

Synthesize

Debrief as a class. Ask students to brainstorm some other functions they could write?

