Applying Functions

Students learn how to apply Functions, and how to interpret the information contained in a Contract: Name, Domain and Range. They then use this knowledge to explore more of the Pyret language.

Prerequisites	Starting to Program
Relevant Standards OK CC-Math	*Select one or more standards from the menu on the left (⌘-click on Mac, Ctrl-click elsewhere).*
Lesson Goals	Students will be able to... • Apply functions to create Images • Identify the parts of a Contract, and use it to apply a function
Student-facing Lesson Goals	• Let's use different types of input to create images with functions.
Materials	• Lesson Slides (Google Slides) • Computer for each student (or pair), with access to the internet • Student workbook, and something to write with
Preparation	• Make sure all materials have been gathered • Decide how students will be grouped in pairs • Make sure student computers can access the Pyret IDE (CPO) • All students should log into CPO and open the "Animals Starter File" they saved from the prior lesson. If they don't have the file, they can open a new one
Supplemental Resources	
Language Table	No language features in this lesson

Glossary

arguments :: the inputs to a function; expressions for arguments follow the name of a function

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

domain :: the type or set of inputs that a function expects

function :: a mathematical object that consumes inputs and produces an output

range :: the type or set of outputs that a function produces

Applying Functions

15 minutes

Overview

Students learn how to apply functions in Pyret, reinforcing concepts from standard Algebra.

Launch

Students know about Numbers, Strings, Booleans and Operators — all of which behave just like they do in math. But what about ***functions***? They may remember functions from algebra: $f(x) = x^2$.

- What is the name of this function?
- The expression $f(2)$ applies the function f to the number 2. What will it evaluate to?
- What will the expression $f(3)$ evaluate to?
- The values to which we apply a function are called its ***arguments***. How many arguments does f expect?

Arguments (or "inputs") are the values passed into a function. This is different from *variables*, which are the placeholders that get *replaced* with input values! Pyret has lots of built-in functions, which we can use to write more interesting programs.

Have students log into CPO and open the "Animals Starter File". If they don't have the file, they can open a new one. Have students type this line of code into the interactions area and hit Enter: `num-sqrt(16)`.

- What is the name of this function?
- What do we think the expression `num-sqrt(16)` will evaluate to?
- What did the expression `num-sqrt(16)` evaluate to?
- Does the `num-sqrt` function produce Numbers? Strings? Booleans?
- How many ***arguments*** does `num-sqrt` expect?

Have students type this line of code into the interactions area and hit Enter: `num-min(140, 84)`.

- What is the name of this function?
- What does the expression `num-min(16, 99)` evaluate to?
- Does the `num-min` function produce Numbers? Strings? Booleans?
- How many ***arguments*** does `num-min` expect?
- What happens if we forget to include a comma between our numbers?

Just like in math, functions can also be *composed* with one another. For example:

```
# take the minimum of 84 and 99, then take the square root of the result
num-sqrt(num-min(84, 99))
```

Investigation

Have students complete Applying Functions (Page 9).

Synthesize

Debrief the activity with the class. What kind of value was produced by that expression? (An Image! New datatype!) Which error messages were helpful? Which ones weren't?

Contracts

35 minutes

Overview

Students learn about **Contracts**, and how they can be used to figure out new functions or diagnose errors in their code. Then they use this knowledge to explore the contracts pages in their workbooks.

Launch

When students typed `triangle(50, "solid", "red")`, they created an example of a new Datatype, called an *Image*.

- What are the types of the arguments `triangle` was expecting?
- How does this output relate to the inputs?
- Try making different triangles. Change the size and color! Try using `"outline"` for the second argument.

The `triangle` function consumes a Number and two Strings as input, and produces an Image. As you can imagine, there are many other functions for making images, each with a different set of arguments. For each of these functions, we need to keep track of three things:

1. **Name** — the name of the function, which we type in whenever we want to use it
2. **Domain** — the type of data we give to the function (names and Types!), written between parentheses and separated by commas
3. **Range** — the type of data the function produces

Domain and Range are *Types*, not specific values. As a convention, we **capitalize Types and keep names in lowercase**. `triangle` works on many different Numbers, not just the `20` we used in the example above!
These three parts make up a ***contract*** for each function. Let's take a look at the Name, Domain, and Range of the functions we've seen before:

```
# num-sqrt :: (n :: Number) -> Number
# num-min :: (a :: Number, b :: Number) -> Boolean
# triangle :: (side :: Number, mode :: String, color :: String) -> Image
```

The first part of a contract is the function's name. In this example, our functions are named `num-sqrt`, and `triangle`.
The second part is the ***Domain***, or the names and types of arguments the function expects. `triangle` has a Number and two Strings as variables, representing the length of each side, the mode, and the color. We write name-type pairs with double-colons, with commas between each one. Finally, after the arrow goes the type of the ***Range***, or the function's output, which in this case is Image.
Contracts tell us a lot about how to use a function. In fact, we can figure out how to use functions we've never seen before, just by looking at the contract! Most of the time, error messages occur when we've accidentally broken a contract.

Investigate

Turn to the back of your workbook, and get some practice reading and using Contracts! Make sure you try out the following functions:

- `text`
- `circle`
- `ellipse`
- `star`
- `string-repeat`

When you've figured out the code for each of these, **write it down in the empty line beneath each contract**. These pages will become your reference for the remainder of the class!
Here's an *example* of another function. Type it into the Interactions Area to see what it does. Can you figure out the contract, based on the example? `string-contains("apples, pears, milk", "pears")`

Possible Misconceptions

Students are *very* likely to randomly experiment, rather than actually using the Contracts page. You should plan to ask lots of direct questions to make sure students are making this connection, such as:

- How many items are in this function's Domain?
- What is the *name* of the 1st item in this function's Domain?
- What is the *type* of the 1st item in this function's Domain?
- What is the *type* of the Range?

Synthesize

You've learned about Numbers, Strings, Booleans, and Images. You've learned about operators and functions, and how they can be used to make shapes, strings, and more!
One of the other skills you'll learn in this class is how to diagnose and fix errors. Some of these errors will be *syntax errors*: a

missing comma, an unclosed string, etc. All the other errors are *contract errors* . If you see an error and you know the syntax is right, ask yourself these two questions:

- What is the function that is generating that error?
- What is the contract for that function?
- Is the function getting what it needs, according to its Domain?

By learning to use values, operations and functions, you are now familiar with the fundamental concepts needed to write simple programs. You will have many opportunities to use these concepts in this course, by writing programs to answer data science questions.
Make sure to save your work, so you can go back to it later!

Additional Exercises:

- Fun with Images
- Reading Contracts
- Matching Expressions and Contracts