Contracts

(Also available for WeScheme)

Students learn how to apply Functions in the programming environment, encounter Image data types, and learn how to interpret the information contained in a Contract: Name, Domain and Range.

Lesson Goals	Students will be able to: • Name and explain the three parts of a Contract • Use Contracts to apply functions that produce Numbers, Strings, and Images • Demonstrate understanding of ***Domain*** and ***Range*** and how they relate to ***Functions***
Student-facing Lesson Goals	• I can make images • I can identify the Domain and Range of a function. • I can use a Contract to apply a function
Materials	• Lesson Slides • Applying Functions (Page 17) • Domain and Range Frayer model (Page 18) • Practicing Contracts: Domain & Range (Page 19) • Matching Expressions and Contracts (Page 20) • Using Contracts (Page 21) • Triangle Contracts (Page 23) • Radial Star (Page 24) • *Optional: Using Contracts (continued) (Page 22)*
Preparation	• Make sure all materials have been gathered • Computer for each student (or pair), with access to the internet • Student workbook, and something to write with • Decide how students will be grouped in pairs • All students should log into code.pyret.org (CPO) and open the "Editor"

Key Points For The Facilitator	• Check frequently for understanding of ***data types*** and ***contracts*** during this lesson and throughout subsequent lessons.
Supplemental Resources	
Language Table	see table below

Types	Functions	Values
Number	`+`, `-`, `*`, `/`, `num-expt`, `num-sqr`, `num-sqrt`	`4`, `-1.2`, `2/3`, `pi`
String	`string-repeat`, `string-contains`	`"hello"`, `"91"`
Boolean	`<`, `<>`, `<=`, `>=`, `<`, `>`, `==`, `<>`, `>=`	`true`, `false`
Image	`star`, `triangle`, `square`	

Click here to see the prior unit-based version.

Glossary

argument :: 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
contract error :: errors where the code makes sense, but uses a function with the wrong number or type of arguments
data types :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure
domain :: the type or set of inputs that a function expects
error message :: information from the computer about errors in code
function :: a mathematical object that consumes inputs and produces an output
name :: how we refer to a function or value defined in a language (examples: +, *, star, circle)
range :: the type or set of outputs that a function produces
syntax error :: errors where the computer cannot make sense of the code (e.g. - missing commas, parentheses, unclosed strings)
variable :: a letter or symbol that stands in for a value or expression

Applying Functions 10 minutes

Overview

Students learn how to apply functions in Pyret , reinforcing concepts from standard Algebra, and practice reading error messages to diagnose errors in code.

Launch

Students know about Numbers, Strings, Booleans and Operators -- all of which behave just like they do in math. But what about ***functions***? Students may remember functions from algebra: $f(x) = x + 4$.

- What is the name of this function? *f*
- The expression $f(2)$ applies the function f to the number 2. What will it evaluate to? *6*
- What will the expression $f(3)$ evaluate to? *7*
- The values to which we apply a function are called its ***arguments***. How many arguments does f expect? *1*

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 code.pyret.org (CPO) , open the editor, type the words **include image** on Line 1 of the Definitions area (left side) and press "Run" to load the image library. Then type `num-sqrt(16)` into the interactions area and hit Enter.

- What is the name of this function? `num-sqrt`
- How many arguments does the function expect? *1*
- What type of argument does the function expect? *Number*
- Does the `num-sqrt` function produce a Number? String? Boolean? *Number*
- What did the expression evaluate to? *4*

Have students type `string-length("rainbow")` into the interactions area and hit Enter:

- What is the name of this function? *string-length*
- How many arguments does `string-length` expect? *1*
- What type of argument does the function expect? *String*

- What does the expression evaluate to? *7*
- Does the `string-length` function produce a Number? String? Boolean? *Number*

Investigation

Have students complete Applying Functions (Page 17) to investigate the `triangle` function and a series of error messages. As students finish, have them try changing the expression `triangle(50, "solid", "red")` to use `"outline"` for the second argument. Then have them try changing colors and sizes!

Synthesize

Debrief the activity with the class.

- What are the types of the arguments `triangle` was expecting? *A Number and 2 Strings*
- How does the output relate to the inputs? *The Number determines the size and the Strings determine the style and color.*
- What kind of value was produced by that expression? *An Image! New data type!*
- Which error messages did you encounter?

Contracts 15 minutes

Overview

This activity introduces the notion of ***Contracts***, which are a simple notation for keeping track of the set of all possible inputs and outputs for a function. They are also closely related to the concept of a *function machine* , which is introduced as well. *Note: Contracts are based on the same notation found in Algebra!*

Launch

When students typed `triangle(50, "solid", "red")` into the editor, they created an example of a new ***data type***, called an *Image* .

The `triangle` function can make lots of different triangles! The size, style and color are all determined by the specific inputs provided in the code, but, if we don't provide the function with a number and two strings to define those parameters, we will get an error message instead of a triangle.

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(s) of data we give to the function
3. **Range** — the type of data the function produces

The ***Name***, ***Domain*** and ***Range*** are used to write a ***Contract***.

Where else have you heard the word "contract"? How can you connect that meaning to contracts in programming?

An actor signs a contract agreeing to perform in a film in exchange for compensation, a contractor makes an agreement with a homeowner to build or repair something in a set amount of time for compensation, or a parent agrees to pizza for dinner in exchange for the child completing their chores. Similarly, a contract in programming is an ***agreement*** *between what the function is given and what it produces.*

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.

Contracts don't tell us *specific* inputs. They tell us the ***data type*** of input a function needs. For example, a Contract wouldn't say that addition requires "3 and 4". Addition works on more than just those two inputs! Instead, it would tells us that addition requires "two Numbers". When we *use* a Contract, we plug specific numbers or strings into the expression we are coding.

Contracts are general. Expressions are specific.

Let's take a look at the Name, Domain, and Range of the functions we've seen before:

A Sample Contracts Table

Name		Domain		Range
`# num-sqr`	::	`Number`	`->`	`Number`
`# num-sqrt`	::	`Number`	`->`	`Number`
`# string-contains`	::	`String, String`	`->`	`Boolean`
`# string-length`	::	`String`	`->`	`Number`
`# triangle`	::	`Number, String, String`	`->`	`Image`

When the input matches what the function consumes, the function produces the output we expect.

Optional: Have students make a Domain and Range Frayer model (Page 18) and use the visual organizer to explain the concepts of Domain and Range in their own words.

Here is an example of another function. `string-append("sun", "shine")`
Type it into the editor. What is its contract? `string-append :: String, String -> String`

Investigate

Have students complete pages Practicing Contracts: Domain & Range (Page 19) and Matching Expressions and Contracts (Page 20) to get some practice working with Contracts.

Synthesize

- What is the difference between a value like `17` and a type like `Number`?
- For each expression where a function is given inputs, how many outputs are there? *For each collection of inputs that we give a function there is exactly one output.*

Exploring Image Functions 20 minutes

Overview

This activity digs deeper into Contracts. Students explore image functions to take ownership of the concept and create an artifact they can refer back to. Making images is highly motivating, and encourages students to get better at both reading error messages and persisting in catching bugs.

Launch

Error Messages

The error messages in this environment are *designed* to be as student-friendly as possible. Encourage students to read these messages aloud to one another, and ask them what they think the error message *means*. By explicitly drawing their attention to errors, you will be setting them up to be more independent in the next activity!

Suppose we had never seen `star` before. How could we figure out how to use it, using the helpful error messages?

- Type `star` into the Interactions Area and hit "Enter". What did you get back? What does that mean? *There is something called "star", and the computer knows it's a function!*
- If it's a function, we know that it will need an open parentheses and at least one input. Have students try `star(50)`
- What error did we get? What *hint* does it give us about how to use this function? `star` *has three elements in its Domain*
- What happens if I don't give it those things? *We won't get the star we want, we'll probably get an error!*
- If I give `star` what it needs, what do I get in return? *An Image of the star that matches the arguments*
- What is the contract for star? *star : Number String String -> Image*

- The contract for `square` also has `Number String String` as the Domain and `Image` as the Range. Does that mean the functions are the same? *No! The Domain and Range are the same, but the function name is different… and that's important because the* `star` *and* `square` *functions do something very different with those inputs!*

Investigate

- At the back of your workbook, you'll find pages with space to write down a contract and example or other notes for every function you see in this course. The first few have been completed for you. You will be adding to these contract pages and referring back to them for the remainder of this Bootstrap class!
- Take the next 10 minutes to experiment with the image functions listed in the contracts pages.
- When you've got working expressions, record the contracts and the code!

(If needed, you can print a copy of these contracts pages for your students.)

Strategies for English Language Learners

MLR 2 - Collect and Display: As students explore, walk the room and record student language relating to functions, domain, range, contracts, or what they perceive from ***error messages***. This output can be used for a concept map, which can be updated and built upon, bridging student language with disciplinary language while increasing sense-making.

Synthesize

- `square` and `star` have the same Domain *(Number, String, String)* and Range *(Image)*. Did you find any other shape functions with the same Domain and Range? *Yes!* `triangle` *and* `circle`.
- Does having the same Domain and Range mean that the functions do the same things? *No! They make very different images!*
- A lot of the Domains for shape functions are the same, but some are different. Why did some shape functions need more inputs than others?
- Was it harder to find contracts for some of the functions than others? Why?
- What error messages did you see? *Too few / too many arguments given, missing parentheses, etc.*

- How did you figure out what to do after seeing an error message? *Read the error message, think about what the computer is trying to tell us, etc.*
- Which input determined the size of the Rhombus? What did the other number determine?

Contracts Help Us Write Code 10minutes

Overview

Students are given contracts for some more interesting image functions and see how much more efficient it is to write code when starting with a contract.

Launch

You just investigated image functions by guessing and checking what the contract might be and responding to error messages until the images built. If you'd started with contracts, it would have been a lot easier!

Investigate

Have students turn to Using Contracts (Page 21), Using Contracts (continued) (Page 22) and use their editors to experiment.

Once they've discovered how to build a version of each image function that satisfies them, have them record the example code in their contracts table. See if you can figure out what aspect of the image each of the inputs specifies. It may help you to jot down some notes about your discoveries. We will be sharing our findings later.

- What kind of triangle did `triangle` build? *The* `triangle` *function draws equilateral triangles*
- Only one of the inputs was a number. What did that number tell the computer? *the size of the triangle*
- What other numbers did the computer need to already know in order to build the `triangle` function? *all equilateral triangles have three 60 degree angles and 3 equal sides*
- If we wanted to build an isosceles triangle or a right triangle, what additional information would the computer need to be given?

Have students turn to Triangle Contracts (Page 23) and use the contracts that are provided to write example expressions. If you are ready to dig into `triangle-sas`, you can also have students work through Triangle Contracts (SAS & ASA).

Sometimes it's helpful to have a contract that tells us more information about the arguments, like what the 3 numbers in a contract stand for. This will not be a focal point of our work, but to give students a taste of it, have them turn to Radial Star (Page 24) and use the contract to help them match the images to the corresponding expressions. For more practice with detailed contracts you can have them turn to Star Polygon to work with the detailed contract for a `star-polygon`. Both of these functions can generate a wide range of interesting shapes!

Synthesize

Make sure that all students have completed the shape functions in their contracts pages with both contracts and example code so they have something to refer back to.

- How was it different to code expressions for the shape functions when you started with a contract?
- For some of you, the word `ellipse` was new. How would you describe what an ellipse looks like to someone who'd never seen one before? Why did the contract for `ellipse` require two numbers? What happened when the two numbers were the same?

How to diagnose and fix errors is a skill we will continue working on developing. Some of the errors are ***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 three 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?

Common Misconceptions

Students are *very* likely to randomly experiment, rather than to actually use the Contracts. 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?

Additional Exercises:

- Matching Images to Code (Desmos)