Contracts

(Also available in Pyret)

Students learn how to apply Functions in the programming environment and interpret the information contained in Contracts: Name, Domain and Range. Image-producing functions provide an engaging context for this exploration.

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	• Let's write code to make images! • Let's learn to identify the Domain and Range of a function. • Let's use Contracts to apply functions.
Prerequisites	• Simple Data Types
Materials	• PDF of all Handouts and Page • Lesson Slides • Printable Lesson Plan (a PDF of this web page)
Supplemental Materials	• Additional Printable Pages for Scaffolding and Practice • *Matching Images to Code (Desmos)*
Key Points For The Facilitator	• Check frequently for understanding of ***data types*** and ***contracts*** during this lesson and throughout subsequent lessons.

Glossary

argument :: the inputs to a function; the expressions for each argument follow the function name

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 type :: a way of classifying values, such as: Number, String, Image, Boolean, or any user-defined data structure

domain :: the type or set of inputs a function expects, i.e., the independent variable(s) that govern the output of the function

function :: a relation from a set of inputs to a set of possible outputs, where each input is related to exactly one output

range :: the type or set of outputs that a function produces, i.e., the dependent variable(s)

syntax error :: errors where the computer cannot make sense of the code (e.g. - missing commas, missing parentheses, unclosed strings)

Applying Functions *10 minutes*

Overview

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

Launch

In human languages, verbs *do things to nouns*. For example, I can "*throw* a ball", or "*eat* a sandwich". "Throw" and "Eat" are verbs, and "ball" and "sandwich" are nouns.

In programming languages, values are like nouns. You've learned about values in our programming language, like Numbers (`42` , `-8.3` , etc), Strings (`"hello!"`), and Booleans (`true` and `false`). In programming, our verbs are called ***functions***. A function is like a machine, and it does three things:

- It takes in some values (called ***arguments***)
- It *does something* to those values
- It produces a new value

Let's play with a few functions, to get the hang of it.

- Log into WeScheme.
- Open the editor and press "Run", then type `(sqrt 16)` into the Interactions Area and hit Enter.
- What part of this expression is the *value*?
  - *16*
- What is the name of this function?
  - `sqrt`
- How many arguments are we giving to this function?
  - *1*
- What is the type of the argument we are giving to `sqrt` ?
  - *A Number*
- What did `sqrt` produce?
  - *4*
- What type of data did `sqrt` produce?

- *Number*

Encourage students to try giving different arguments to `sqrt`. Does it only work with Numbers? Does it only take one Number?

- 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*
- What type of data did `string-length` produce?
  - *Number*

Encourage students to try giving different arguments to `string-length`. Does it only work with Strings? Does it only take one String? What does it do?

Investigation

- Complete the first section of Applying Functions to investigate the `triangle` function.
- Try changing the expression `(triangle 50 "solid" "red")` to use `"outline"` for the second argument. Now try changing colors and sizes!
- Now, take a look at some buggy code in the bottom section. Can you spot the mistakes?

Synthesize

Debrief the activity with the class. Be sure to discuss and analyze different error messages encountered.

- 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!*

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
`; +`	:	`Number, Number`	->	`Number`
`; -`	:	`Number, Number`	->	`Number`
`; /`	:	`Number, Number`	->	`Number`
`; *`	:	`Number, Number`	->	`Number`
`; sqr`	:	`Number`	->	`Number`
`; sqrt`	:	`Number`	->	`Number`
`; <`	:	`Number, Number`	->	`Boolean`
`; >`	:	`Number, Number`	->	`Boolean`
`; <=`	:	`Number, Number`	->	`Boolean`
`; >=`	:	`Number, Number`	->	`Boolean`
`; ==`	:	`Number, Number`	->	`Boolean`
`; <>`	:	`Number, Number`	->	`Boolean`
`; string=?`	:	`String, String`	->	`Boolean`
`; string-contains?`	:	`String, String`	->	`Boolean`
`; string-length`	:	`String`	->	`Number`
`; triangle`	:	`Number, String, String`	->	`Image`

- What do you Notice?
- What do you Wonder?

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 and use the visual organizer to explain the concepts of Domain and Range in their own words. You might also have students complete Function and Variable Frayer model.

- 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`
- What do you think `string-append` does?
  - It links together two different strings.

Investigate

Complete Practicing Contracts: Domain & Range and Matching Expressions and Contracts to get some practice working with Contracts.

Synthesize

- What is the difference between a value like `17` and a type like `Number`?
  - *A value is a specific piece of data, whereas a type is a way of classifying values.*
- 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

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

> **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!

- 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. 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

Today's lesson will focus on these image-producing functions. If you're using a printed workbook with your class, a list of all of the functions used in this course can be found in the back of the book, along with space to write down a Contract and example or other notes for each of them.

- Turn to Contracts for Image-Producing Functions and take the next 10 minutes to experiment with the functions.
- When you've got working expressions, record the contracts and the code!

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

- Does having the same Domain and Range mean that two functions do the same things?
  - *No! For instance, `square`, `star`, `triangle` and `circle` all have the same Domain and Rnage, yet 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? How did you figure out what to do after seeing an error message?
  - *Error messages include: too few / too many arguments given, missing parentheses, etc. Reading the error message and thinking about what the computer is trying to tell me can inform next steps.*

- Which input determined the size of the Rhombus? What did the other number determine?

Contracts Help Us Write Code *10 minutes*

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

- Complete Using Contracts, experimenting with your editor.
- *Optional:* Try Using Contracts (2) for additional practice with contracts.

Once students have discovered how to build a version of each image function that satisfies them, have them record the example code in their contracts table. Encourage students to explore what aspect of the image each of the inputs specifies. It may help students to jot down notes about their discoveries.

- 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?
  - *A right triangle requires the base (Number) and the height (Number). An isosceles triangle requires a leg (Number) and an angle (Number).*
- Now, turn to Triangle Contracts and use the contracts that are provided to write example expressions.

Optional: If students are ready to dig into more complex triangles, you can also have them work through Triangle Contracts (SAS & ASA).

Turn to Radial Star and use the provided Contract to help you match the images to the corresponding expressions.

Contracts that tell us more information about the arguments can be helpful, but are not a focal point of our work in this course. The Radial Star worksheet is designed to give students a taste of these more complex contracts. *Optional:* For more practice with detailed contracts, students can also complete Star Polygon. Both `star-polygon` and `radial-star` generate a wide range of interesting shapes!

Make sure that all students have added contracts and example codes to Contracts for Image-Producing Functions so they have something to refer back to.

Students as Teachers

It can be empowering for students to develop expertise on a topic and get to share it with their peers! This section of the lesson could be reframed as an opportunity for students to become experts in an image-producing function and teach their classmates about it. For example, Pair 1 and pair 4 might focus on `radial-star`, pair 2 and pair 5 might focus on `polygon-star`, pair 3 and pair 6 might focus on `regular-polygon`, etc. First, each pair would explore their function. Perhaps each pair could make a poster, starter-file or slide deck about their function including: the Contract, an explanation of how it works in their own words, a few images that it can generate illustrating the range of possibilities with the expressions that generate them. Next, they might compare their thinking with another pair that focused on the same Contract. Finally, pairs could be grouped with other pairs who focused on different functions and teach each other about what they learned.

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?

Synthesize

- 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 has never seen one before?
- Why did the Contract for `ellipse` require two numbers? What happened when the two numbers were the same?

Diagnosing and fixing errors are skills that students will continue developing throughout this course. Some of the errors are ***syntax errors***: a missing comma, an unclosed string, etc. All 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?

Additional Exercises

- Matching Images to Code (Desmos)