**Unit 2 – Reading 1**

**Writing Functions in Pyret**

**Functions and Computation**

Computers are powerful tools used by scientists. One of the primary ways we will communicate with computers in this course is by writing functions. In coding, a function is an instruction we write for a computer which it follows to transform some input or inputs into a certain output. This makes a function ideal for anything we might want the computer to do many times.

**Function Contracts**

Before we can start writing the instructions of our function, we must be sure of what exactly the function is supposed to accomplish. For this, we need only know what we will call our function, what inputs we will need to provide, and what kind of output we should receive after the computer runs our function. The exact procedures the computer will follow to get from the inputs to the output is not important at this step.

We call this initial information the function’s **contract**, and we write them as follows

|  |
| --- |
| function-name :: (Input-identifier :: Input-type) -> Output-type |

Remember that :: means “is defined as” in Pyret. Let’s look at some examples of function descriptions and the contracts we would write for those functions.

1. We want to write a function called add-five which consumes a single Number x as an input and produces a new Number which is five times larger than x.

|  |
| --- |
| add-five :: (x :: Number) -> Number |

1. We want to write a function called number-of-sides which consumes an Image of a shape and produces the Number of sides that shape has.

|  |
| --- |
| number-of-sides :: (shape :: Image) -> Number |

1. We want to write a function called name-of-shape which consumes a Number called sides and produces a String of the name of the polygon with that number of sides.

|  |
| --- |
| name-of-shape :: (sides :: Number) -> String |

1. We want to write a function called day-of-the-week which consumes three Strings called day, week, and month and produces a String of the name of the day of the week.

|  |
| --- |
| day-of-the-week :: (day :: String, week : String,  month :: String) -> String |

**Choosing your Inputs**

One important aspect of writing a new function is deciding what information needs to be included as an input to your function. Take the following situation as an example:

|  |
| --- |
| A student working a part time job makes $11 an hour. She wants to write a function to determine how much money she will make if she works various numbers of hours. |

The student in the example above writes two possible contracts for her function, which she will call amount-earned, and she must decide which one she wants to use.

Contract #1

|  |
| --- |
| amount-earned1 :: (hours-worked :: Number) -> Number |

Contract #2

|  |
| --- |
| amount-earned2 :: (hours-worked :: Number, hourly-pay ::  Number) -> Number |

In order to choose between these two contracts, the student needs to ask herself the following question: **“What values are changing each time I run this function?”**

For the student making $11 an hour, her hourly pay is the same every time she would use this function, while the number of hours she might work will be different. Thus, in this instance, the function amount-earned1 would be more appropriate for her situation.

However, imagine that student wants to be able to use her amount-earned function to predict how much money all her friends will make from their jobs, which all pay somewhere between $10 and $15 an hour. In that case it’s not just the number of hours worked, but also the hourly pay that will change each time she uses her function. Thus, she would have to use the amount-earned2 function.

From this we can see that the intended use of a function determines what parameters it must take as inputs.

**Writing Examples**

Once we have written a contract for our function, the next step is to come up with a few examples of our function working properly. These examples serve two purposes:

1. The help us identify a pattern between our inputs and outputs which can help us write the instructions for our function down the line.
2. They serve as a test to see whether our finished function behaves the way we expect it to.

Since we may not always know what pattern we expect our function to follow, coming up with examples of how our function should behave ***before*** writing it will help us figure out exactly how to write our function.

Let’s say we want to write a function called find-difference which consumes two Numbers, x and y, and produces a Number which is the difference between them. Below is the contract and some examples for this function.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8. | find-difference :: (x :: Number, y :: Number) -> Number  **examples:**  find-difference(10, 10) **is** 0  find-difference(-5, -15) **is** 10  find-difference(20, 10) **is** 10  find-difference(10, 20) **is** 10  **end** |

These examples tell us something important about how we want our function to behave. The first, as you might expect, is that the difference between a number and itself is zero. The second is that we can use our function to find the difference between positive and negative numbers. The third, and possibly most surprising, is that the difference between two numbers is the same regardless of the order in which we put them into our function. We can also see that the difference between two numbers is positive in all of our examples, which gives us a clue about how we should write this function.

**Writing Functions**

Once we have our examples written, we are ready to write the actual instructions which the computer will follow every time we use our function. Let’s take a simple function called find-sum, which consumes two Numbers x and y and produces a Number which is the sum of x and y. Let’s see what this looks like in Pyret.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | find-sum :: (x :: Number, y :: Number) -> Number  **examples:**  find-sum(0, 5) **is** 5  find-sum(-5, 5) **is** 0  find-sum(5, 5) **is** 10  **end**  **fun** find-sum(x, y):  x + y  **end** |

The actual instructions for carrying out this function are found on lines 9-11 in the block above. Notice the notation we use to tell Pyret we are giving it instructions for how to carry out a function:

|  |
| --- |
| **fun** function-name(input-identifiers):  instructions  more-instructions (if necessary)  **end** |

Once we have written the instructions tell Pyret how to carry out a specific function, we are free to use it as many times as we want.

**Design Recipe**

One tool we use as programmers when designing new functions is the **design recipe**. This walks us through all the steps we saw in this reading: Writing a contract, coming up with examples, then writing the instructions for a function.

You should always fill out a design recipe prior to writing a new function. Think of a design recipe as a ***function proposal***, which must be approved before construction on a new function can begin. Look over the sample design recipe included at the end of this reading.

**Try for Yourself**

Open the Pyret program found below and work your way through the sample functions. Fill in any missing information (denoted by a … in the code). Once you’ve worked your way through all the examples, try writing a function of your own from scratch, including its contract and examples.

Code: <https://tinyurl.com/yc554898>

|  |
| --- |
| **Design Recipe** |

|  |
| --- |
| **Physical Interpretation** |

What will the input(s) of your function be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: length)

What will the units of each input be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: meters)

What will the output be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: density)

What will the unit of the output be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: grams/meter3)

|  |
| --- |
| **Contract & Purpose Statement** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # |  | :: |  | -> |  | |
|  | name |  | Domain (type of input(s)) |  | Range (type of output) | |
| # |  | | | | | |
|  | What does the function do? (The function consumes \_\_\_\_\_\_ and produces \_\_\_\_\_\_\_.) | | | | | |
| **Examples** | | | | | |

Write examples of your function in action

**examples:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

name example input(s)

**is**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What calculation must be performed?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

name example input(s)

**is**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What calculation must be performed?

**end**

|  |
| --- |
| **Function Definition** |

Circle the changing quantities in your examples and name them (consider the names used for the physical quantities above).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **fun** |  | ( |  | ): |
|  |  | | | |
|  |
| **end** |  |  |  |  |

|  |
| --- |
| **Design Recipe** |

|  |
| --- |
| **Physical Interpretation** |

What will the input(s) of your function be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: length)

What will the units of each input be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: meters)

What will the output be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: density)

What will the unit of the output be?\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(ex: grams/meter3)

|  |
| --- |
| **Contract & Purpose Statement** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # |  | :: |  | -> |  | |
|  | name |  | Domain (type of input(s)) |  | Range (type of output) | |
| # |  | | | | | |
|  | What does the function do? (The function consumes \_\_\_\_\_\_ and produces \_\_\_\_\_\_\_.) | | | | | |
| **Examples** | | | | | |

Write examples of your function in action

**examples:**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

name example input(s)

**is**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What calculation must be performed?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

name example input(s)

**is**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

What calculation must be performed?

**end**

|  |
| --- |
| **Function Definition** |

Circle the changing quantities in your examples and name them (consider the names used for the physical quantities above).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **fun** |  | ( |  | ): |
|  |  | | | |
|  |
| **end** |  |  |  |  |