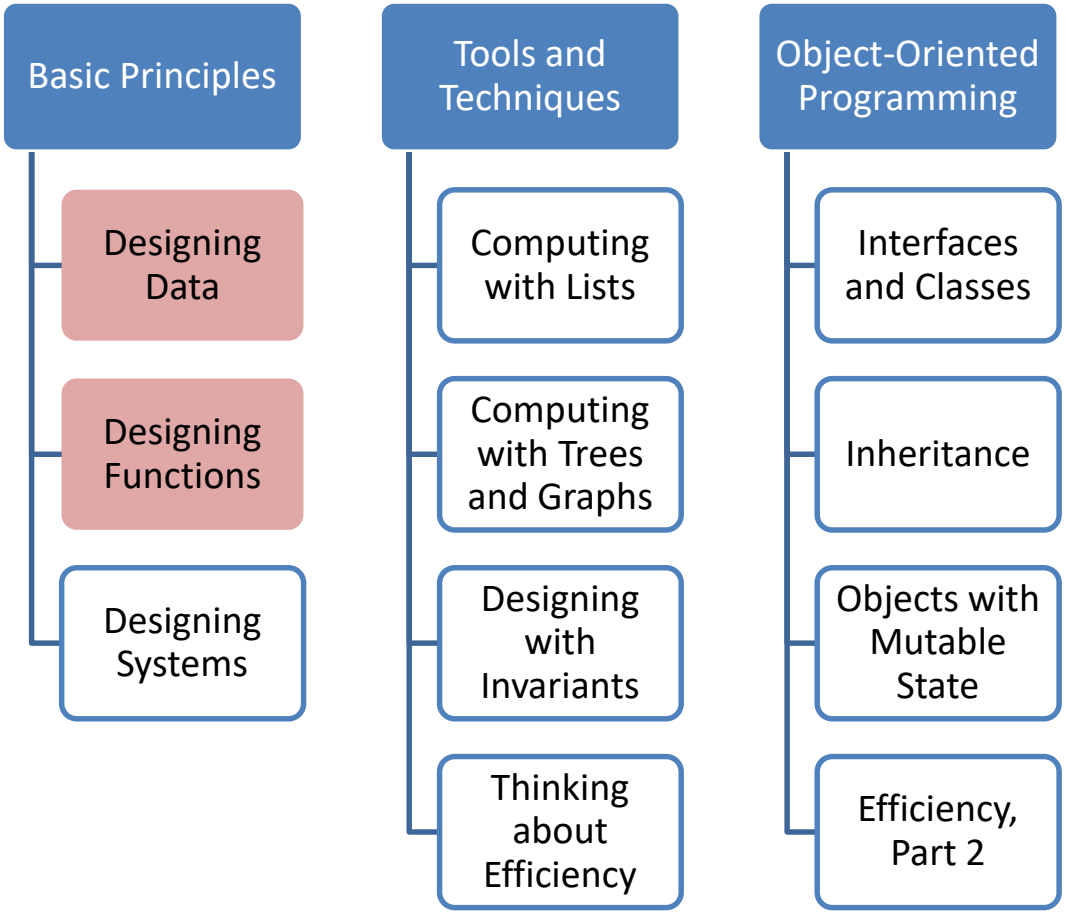
The Function Design Recipe

CS 5010 Program Design Paradigms "Bootcamp"

Lesson 1.1

Module 01


Basic Principles
Designing Data
Designing Functions
Designing Systems
Tools and Techniques
Computing with Lists
Computing with Trees and Graphs
Designing with Invariants
Thinking about Efficiency
Object-Oriented Programming
Interfaces and Classes
Inheritance
Objects with Mutable State
Efficiency, Part 2

Learning Objectives

- By the time you complete this lesson, you should be able to:
  - list the 6 steps of the Function Design Recipe
  - briefly explain what each step is.
  - explain the difference between information and data, and explain the role of representation and interpretation.

The function design recipe

- The function design recipe is the most important thing in this course. It is the basis for everything we do.
- It will give you a framework for attacking any programming problem, in any language. Indeed, students have reported that they have found it useful in other courses, and even in their everyday life.
- With the recipe, you need never stare at an empty sheet of paper again.
- Here it is:

The Function Design Recipe

The Function Design Recipe
1. Data Design
2. Contract and Purpose Statement
3. Examples and Tests
4. Design Strategy
5. Function Definition
6. Program Review

This is important. Write it down, in your own handwriting. Keep it with you at all times. Put it on your mirror. Put it under your pillow. I'm not kidding!

Brief Explanation of the Recipe

- **Data Design**: what kind of data does your system deal with, and what does each possible value of the data mean?
- **Contract**: what kinds of values does your function take as its arguments, and what kind of values does it return?
- **Purpose Statement**: given a particular input, which value should the function return?
- **Examples/Tests**: what is a typical call of your function? How can somebody tell whether your function is returning a correct value?
- **Strategy**: how does your function compute the desired value? Describe the way it works in a tweet.
- **Function Definition**: the code of the function.
- **Program Review**: now that you have a working function, how can it or its explanation be improved to make it clearer to a reader?

A Function Designed According to the Recipe

```
;; DATA DEFINITIONS:
;; A FarenTemp   is represented as a Real.
;; A CelsiusTemp is represented as a Real.

;; f2c: FarenTemp -> CelsiusTemp
;; GIVEN: a temperature in Fahrenheit,
;; RETURNS: the equivalent temperature in Celsius

;; EXAMPLES:
;; (f2c 32) = 0
;; (f2c 212) = 100
;; DESIGN STRATEGY: Transcribe Formula

(define (f2c x)
  (+ (* 5/9 x) -160/9))

;; TESTS
(begin-for-test
  (check-equal? (f2c 32) 0
    "32 Fahrenheit should be 0 Celsius")
  (check-equal? (f2c 212) 100
    "212 Fahrenheit should be 100 Celsius"))
```

Data Definitions: What real-world we are representing, and how they are represented.

Contract (or signature): what kinds of values the function takes as arguments, and what kind of value it returns

Purpose Statement: given a particular input, what is the value that the function should return?

Examples (for the reader)

Design Strategy: Brief description of how the function gets the answer.

Function Definition

Tests (executable)

The recipe is a recipe

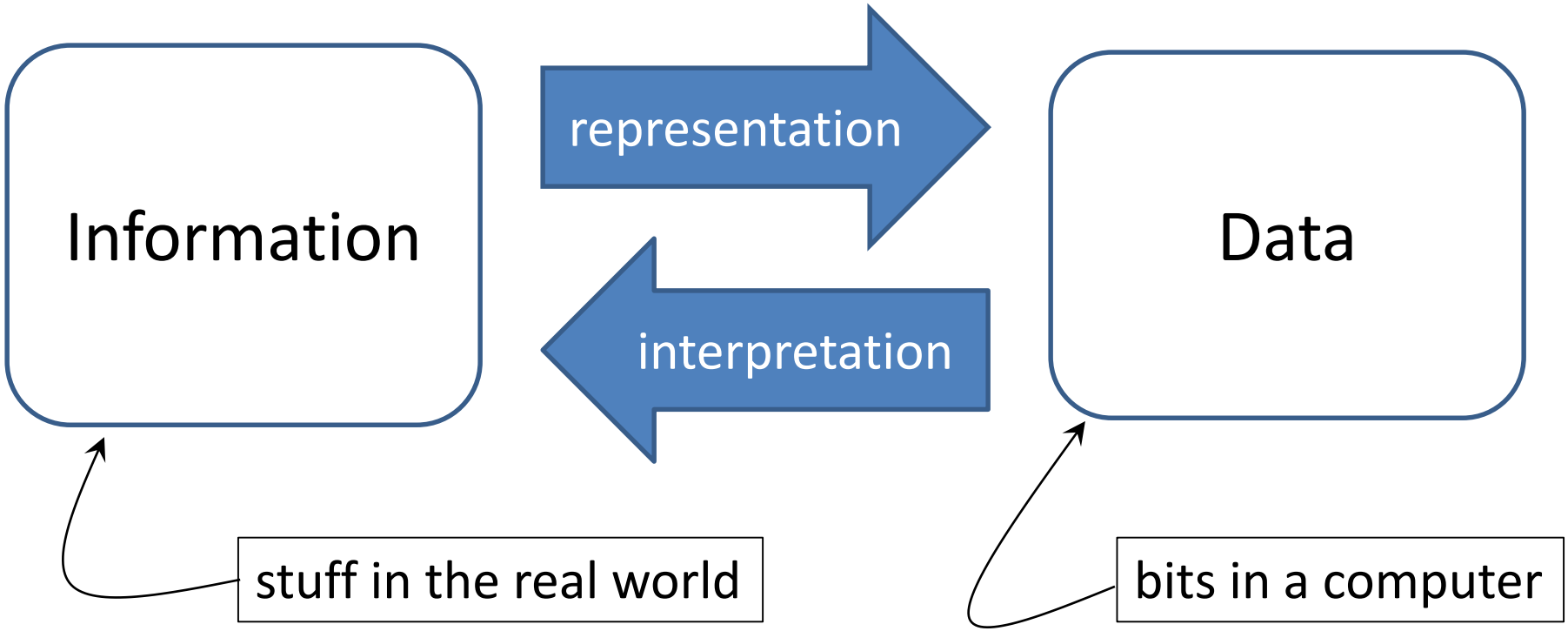
- It's not just a list of components
- It tells you the *order* in which you should do them.
- Each step depends on the preceding ones.
- If you do them out of order, you *will* get in trouble (trust me!)

In the rest of this lesson, we will discuss each step in turn.

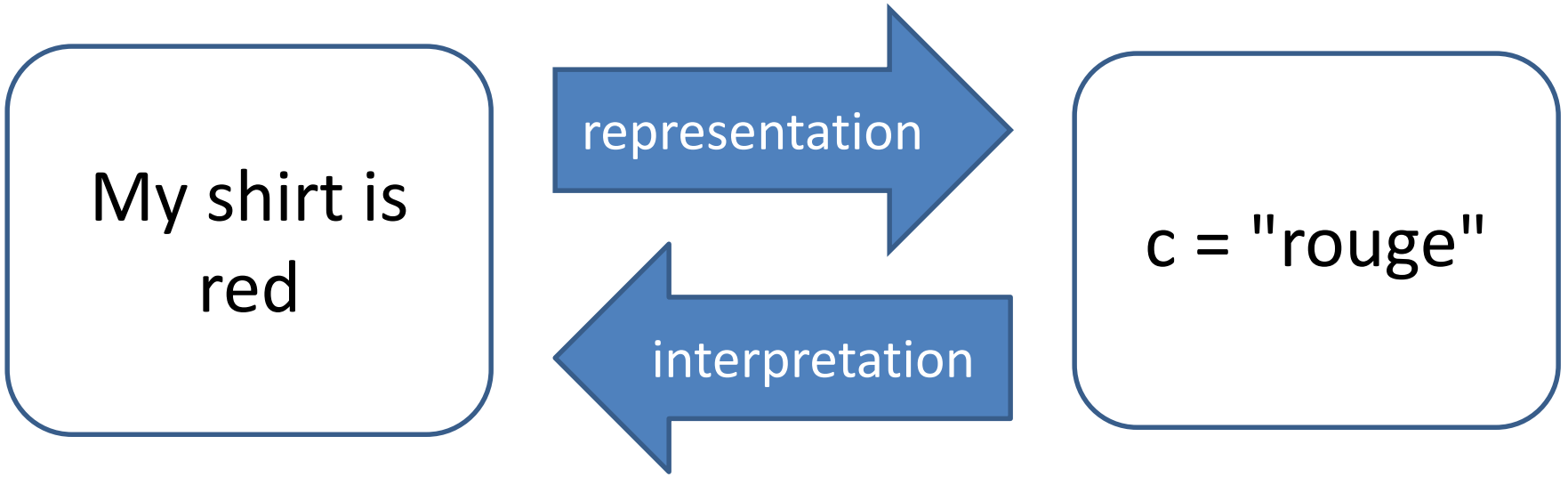
Step 1: Information Analysis and Data Design

- Information is what lives in the real world. To do this step, you need to do 3 things:
  1. You need to decide *what part* of that information needs to be represented as data.
  2. You need to decide *how* that information will be represented as data
  3. You need to document how to *interpret* the data as information

The relation between information and data



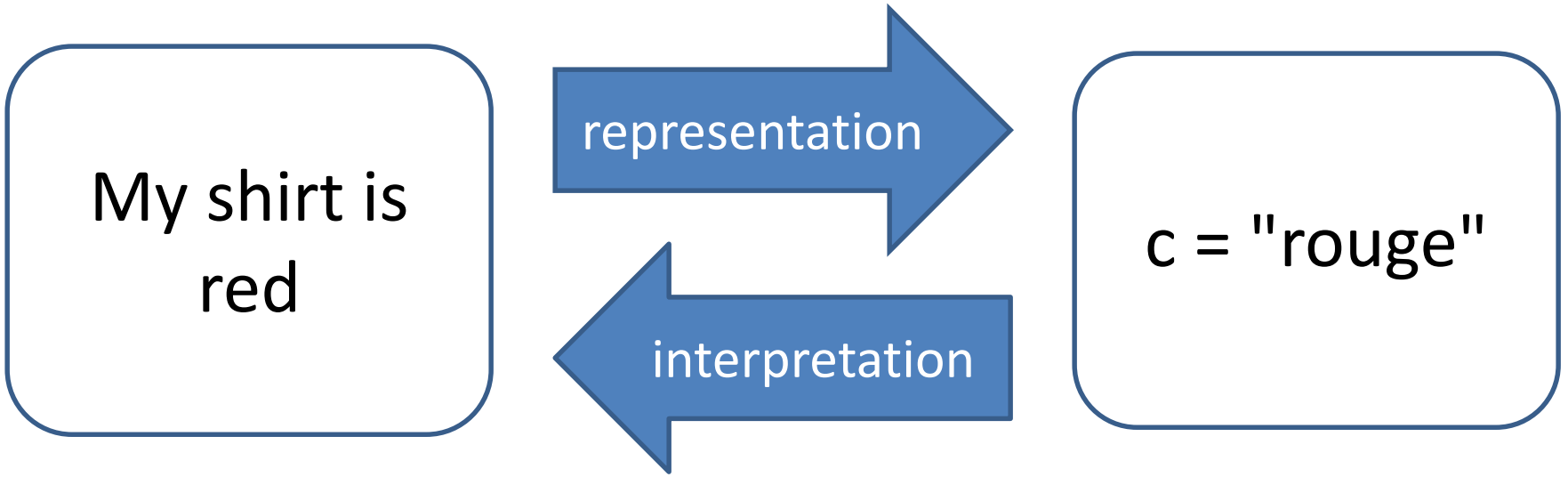
Information and Data: Example



How do we know that these are connected?

Answer: we have to write down the interpretation.

Information and Data: Example



Interpretation:

c = the color of my shirt, as a string, in French

This is part of the program *design* process.

Deliverables for Step 1 (Information Analysis and Data Design)

1. Description of the real-world information to be represented.
2. Structure Definitions: declarations of new data structures if any.
3. Constructor Template: a recipe for building values of this data type.
4. Interpretation: what each value of the type represents.
5. Observer Template: a template for functions that look at values of this data type.
6. Examples (if needed so reader will understand)

That first step was a big one!

- ... but important: the vast majority of errors in student programs can be traced back to errors in step 1!
- We'll go through each of these steps in more detail in Lessons 1.3-1.5

Step 2: Contract and Purpose Statement

- *Contract:* specifies the kind of input data and the kind of output data
- *Purpose Statement:* A set of short noun phrases describing *what* the function is supposed to return. These are typically phrased in terms of information, not data.
  - The generally take the form GIVEN/RETURNS, where each of these keywords is followed by a short noun phrase.
  - When possible, they are phrased in terms of information, not data.

Examples of Contract and Purpose Statements

```
;; f2c: FarenTemp -> CelsiusTemp
;; GIVEN: a temperature in Fahrenheit,
;; RETURNS: the equivalent temperature in
Celsius

;; f2mars : FarenTemp -> CelsiusTemp
;; GIVEN: Any temperature in Fahrenheit
;; RETURNS: The mean temperature on the surface
;;   of Mars, in Celsius
```

Examples of Contract and Purpose Statements (2)

```
scene-with-cat : Cat Scene -> Scene
GIVEN: a Cat c and a Scene s
RETURNS: A Scene like s, except that the Cat c
has been painted on it.
```

Of course there are no cats in our computer. What this means is:
c is the representation of some cat,
s is the representation of some scene,
and the function returns a representation of a scene like the one **s** represents, except the new scene contains an image of the cat.

Step 3: Examples and Tests

- Examples: some sample arguments and results, to make clear to the reader what is intended.

```
;; (f2c 32) = 0
;; (f2c 212) = 100
```

Tests

- Unlike examples, tests are meant to be executable. Your tests will live in the file with your code, so they will be run every time you load your file. That way if you inadvertently break something, you'll find out about it quickly.
- Our testing framework is based on rackunit and allows the tests to appear anywhere in the file; they are executed at the end of the file. You should try to put the tests near the function they test– see the example files.

Tests (2)

Here are the tests we wrote for **f2c**. Since we know that **f2c** must be a linear function, two tests suffice to guarantee that we got the constants right.

```
(begin-for-test
 (check-equal? (f2c 32) 0
   "32 Fahrenheit should be 0 Celsius")
 (check-equal? (f2c 212) 100
   "212 Fahrenheit should be 100 Celsius"))
```

Step 4: Design Strategy

- A short description of how to get from the purpose statement to the function definition
- We will have a menu of strategies.
- We'll cover this in more detail in Module 2

Here is our starting list of strategies:

There will be more…

Design Strategies
1. Transcribe formula
2. Use template for <data def>
3. Divide into cases on <condition>

Design Strategy for f2c

- For f2c, the strategy we used was “transcribe formula”
  - this is, we wrote down the mathematical function and transcribed into Racket.

Step 5: Function Definition

To define our function, we apply some external knowledge. We know that Fahrenheit and Celsius are related linearly, so the solution must be of the form

$$f2c(x) = ax + b.$$

So we take our two examples and get two simultaneous equations:

$$x = 0: \quad 32a + b = 0$$
$$x = 212: \quad 212a + b = 100$$

We solve for a and b, getting

$$a = \frac{5}{9}, b = -\frac{160}{9}$$

Function Definition

- Now we can write the code.
  - Our code is just a transcription of the formula into Racket, using the fact that Racket has rational numbers.

```
(define (f2c x)
  (+ (* 5/9 x) -160/9))
```

Step 6: Program Review

- Did the tests pass?
- Are the contracts accurate?
- Are the purpose statements accurate?
- Can the code be improved?

Summary

- In this lesson, we have learned the steps of the Function Design Recipe.
  - 6 steps
  - You need to do them *in order*.
  - The Design Recipe gives you a plan for attacking any programming problem
  - It is the single most important thing in this course!!

Next Steps

- Review 01-1-f2c.rkt in the Examples folder.
  - Download and run it. Make some changes. What happens when you change the file? What kinds of error messages do you get?
- If you have questions about this lesson, post them on Piazza.
- Go on to the next lesson.