

CSC151.02 2014S: Extra Session 3

Overview

- Admin
- Q and A

Admin

Questions

How do I run this stuff on my Mac or PC?

Wait a few days. Sam is getting a virtual computer ready for you to use. It will be a little slower, but it should work.

Could you talk more about the testing stuff?

- Problem when we write procedures: How do we know that the procedures we write are correct?
- Multiple approaches
 - Formal proof (like mathematicians)
 - Run lots of experiments (like scientists)
- Note: Great in theory, mixed in practice.
 - More tests tend to lead to better code.
- Example: We've written a procedure that computes a quadratic root. How do we know it's correct? We try it on a few things that we know the answer to.
 - $(x - 2)(x + 3) = x^2 + x - 6$
- What's the difference between failure and error?
 - Failure: Gave the wrong result
 - Error: Crashed (e.g., because of wrong input)
- Test-Case vs. Test-Suite
 - Test case: One test (or one small set of tests), are evaluated instantly by DrRacket.
 - Doesn't need a define
 - Test suite: A group of test cases, potentially run later
 - Can be nested
 - Run with (run-tests NAME-OF-SUITE)
 - Needs a define.
 - Important: You can choose when to run the tests.
- For the exam, I'm okay if you just use check-... or test-case.

Sam could not capture all of what he said on the eboard. Here's the code we ended up with.

```
#lang racket
(require rackunit)
(require rackunit/text-ui)

(define root1
  (lambda (a b c)
    (/ (- (- b)
          (sqrt (- (* b b) (* 4 a c))))
        (* 2 a))))

(define root2
  (lambda (a b c)
    (/ (+ (- b)
          (sqrt (- (* b b) (* 4 a c))))
        (* 2 a))))

(define root1-tests
  (test-suite "testing root1"
    (test-case "simple roots"
      (check-equal? (root1 1 1 -6) -3)
      (check-equal? (root1 1 -2 1) 1))))

(define root2-tests
  (test-suite "testing root2"
    (test-case "simple roots"
      (check-equal? (root2 1 1 -6) 2)
      (check-equal? (root2 1 -2 1) 1))))

(define root-tests
  (test-suite "Testing both roots"
    root1-tests
    root2-tests))
```

What will be on the quiz?

Quizzes typically have material from the past Friday->Wednesday/Tuesday

So ... documentation, testing, and side-effecting procedures.

Documentation: Likely to ask for preconditions or postconditions, or provide part and ask for the rest.

Testing: Ask what you would test. (Syntax not important. Thinking about what good "experiments" would be is important.) We will not expect you to know the weirdo `check-exn` thing.

Side-effecting procedures. Write a procedure that uses `display`, `newline`, `image-load`, `image-save`, `drawing-render!`, `image-new`, `image-show`.

Will you include the drawing and arithmetic procedures on the quiz?

Probably.

When writing preconditions, how often can we write "no additional"?

If you are confident that your descriptions of the parameters suffice for specifying all legal inputs.

`image-load` probably has a parameter of "filename, a string" and the preconditions are things like "the file referenced by filename exists; the file referenced by filename contains an image in jpg, png, or ... format; the user running this program has permission to access that file."

```
;;; Procedure:
;;; image-load
;;; Parameters:
;;;   fname, a string
;;; Purpose:
;;;   Load the image stored in the file named by fname.
;;; Produces:
;;;   image, an image id
;;; Preconditions:
;;;   fname names a file that contains an image.
;;;   That image is one of the standard formats (e.g., PNG, JPG).
;;;   The suffix of the file matches the image type.
;;;   The person executing the code has permission to access the file.
;;; Postconditions:
;;;   (image-show image) will produce a rendering of the given file.
```

If we take more than five hours, can we still invoke "There's more to life?"

Yes. But if it's taking you much more than five hours, you are better off stopping and talking to me later so that you'll do better on the next exam.

I think of drawing-render! as "put this drawing on top of an existing image". Is that a correct interpretation?

Yes.

Samuel A. Rebelsky, rebelsky@grinnell.edu

Copyright (c) 2007-2014 Janet Davis, Samuel A. Rebelsky, and Jerod Weinman. (Selected materials are copyright by John David Stone or Henry Walker and are used with permission.)



This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.