Lab: Welcome to Beginning Student Language and DrRacket (Here Are Your Errors)

COMP 50

Fall 2013

Prelude

The purpose of this lab is to prepare you to work independently with the Beginning Student Language and with DrRacket.

Vocabulary and Background: Beginning Student Language is a *programming language*. Other examples of programming languages include Python and C++. DrRacket is a *programming environment*. Other examples of programming environments include Eclipse and Visual Studio. All these programming environments are *interactive development environments*, usually abbreviated *IDEs*. Programming languages are cheap to build; IDEs are much more expensive.

Overview

Beginning Student Language provides a uniform syntax [1] for the sorts of math you might have studied in a pre-algebra course, plus a collection of predefined functions like `expt` and `log`.

In a pre-algebra course where you work on paper you can write anything you want. This means that you can write down things that aren't even formulas, and the first time you'll know there's a problem is when you get your paper back and it is bleeding red ink. But when you write expressions and function definitions in Beginning Student Language, DrRacket checks them right away, and if they are *not* well formed, you get an *error message*. All beginners make mistakes; an error message is DrRacket's way of letting you know that you've made a beginner's mistake. Nobody likes getting an error message, but for a programmer, error messages help: you can fix your mistake early, without wasting a lot of time building on something that is wrong.[2] A key outcome of this lab is that by the time you finish, you must have seen error messages, you must know how to read them, and you must know how to react to them.

In this lab, you will experiment with DrRacket's three most important tools:

1. the Definitions area, where you define functions, state tests, and write down expressions your program must evaluate
2. The Interactions area, where you ask DrRacket to evaluate expressions and to experiment with ideas
3. The Stepper, which helps you step through the evaluation of function definitions and expressions from the Definitions area

When you walk out of the lab, you must feel comfortable using all three tools.

> *The best programmers, like the best craftsmen, understand the tools they use.* —Jack W. Davidson

[1] **Vocabulary**: *Syntax* is a set of rules that govern or explain how something is written. It is a work shared with computer science and linguistics. Programming-language syntax says what utterances constitute well-formed programs. Natural-language syntax says what utterances constitute well-formed (grammatical) sentences. (Note: *well-formed* is different from *meaningful*.) The syntax of Beginning Student Language is described in Intermezzo I of your book. You will study it later.

[2] Hint: Any time you are designing and building software, you will do better if you develop the habit of clicking Check Syntax and Run *early* and *often*. This habit carries through to many other classes.

Structure of the lab

When you enter the lab, hand in your signed copy of the Bargain to the copilot TA. The copilot will match students in pairs.

Pair programming

During most labs you will work with a partner. In the partnership, one of you is the pilot and the other is the copilot The pilot types at the keyboard while the copilot looks over the pilot's shoulder. During the lab, as during an airplane flight, pilots and copilots switch back and forth.

Launching DrRacket at your own computer

Launch DrRacket. If you're lucky, DrRacket will be in the Applications menu under Programming. Otherwise you will probably need to open a Terminal window and type `drracket`.

Choose the Beginning Student Language from the {Language | Choose Language} drop-down menu.

Interacting with DrRacket

[45 min]

The lab teacher will use DrRacket's interactions area as a calculator. You'll see a range of operations on numbers, booleans, strings. You'll see big numbers and complex numbers. In the future, you don't need a calculator; use the interactions area.

> It is now your time to play with a partner.

The lab leader will use DrRacket's definitions area to write down expressions. Nothing happens! Now it's time for Run and for Step.

> Your time to play again. Define the following function in the Definitions area:
>
> - The function `how-far` consumes the number of minutes you drive at 67mph and produces how far you get in the given time.
>
> Run. Interact. Add an application of how-far to the definitions area. Use the stepper.
>
> - Define a function that accepts a number of minutes and computes how many whole hours these minutes represent.

You've Got Errors

When you program, you encounter three kinds of errors. From least difficult to most difficult, we classify them as follows:

- *Syntax errors* manifest as red text from DrRacket. They mean that what you wrote is not a "sentence" in the Beginning Student Language. You can trigger a syntax error by clicking Check Syntax.

  Example of a syntax error in English: "fish green unfurble right."

- *Run-time errors* manifest as red text from DrRacket. They mean that you wrote a Beginning Student Language sentence, but when you try to interact with it, something meaningless or bad happened. For example, a function was applied to too many or too few arguments or to the wrong kinds of arguments, or the code tried to test a condition that was not Boolean, or any of a plethora of other things. If you have a run-time error, your program doesn't compute a final result.

  Classic analog of a run-time error in English (Noam Chomsky, 1957): "Colorless green ideas sleep furiously." The sentence is grammatical, but it has no operational meaning.

- *Logical errors* manifest as *perfectly plausible wrong answers*. If you have a logical error, your program computes something but it's wrong.

  As an analog of a logical error in English, consider these directions to Halligan Hall: "Go down Boston Avenue toward West Medford and stop just before Alewife Brook Parkway." This sentence is grammatical and it has a perfectly good operational meaning, but it doesn't get you to the right place.

Syntax errors and run-time errors you can learn to deal with. To defined yourself against logical errors, you must use `check-expect`, `check-within`, and `check-error`.

The lab teacher will call for a solution to `how-far` and will demonstrate all three kinds of errors.

Switch partners

Conditions

The lab teacher will demonstrate a conditional function:

> The function `how-hot` consumes a temperature (number) and produces one of three strings: "cold" for temperatures below 45 (inclusive), "comfortable" for temperatures between 45 (exclusive) and 75 (inclusive), and "hot" for temperatures above 75.

The lab teacher will explain `cond` and will explain the use of square brackets. Ask the lab teacher to use Step. Beware the stepper and `check-within`—it skips a step!

It is your turn.

> Define the function `letter-grade`. It consumes a score (number) between 0 and 100 and produces a letter grade ("A", "B", "C", "D", or "F"). Choose your own grading scale.

Stretch problem: Conversion to English units for handheld GPS

The Garmin eTrex handheld GPS renders distances in three ways:

- Any distance up through 999 feet is rendered to the nearest whole foot, followed by the abbreviation "ft".
- Any distance from 1000 feet up through 999 miles is rendered to the nearest *tenth* of a mile, using a decimal point and the abbreviation "mi".
- Any distance of 1000 miles or more is rendered to the nearest mile, with no decimal point and the abbreviation "mi".

Implement function `meters->english` which converts a distance in meters to a string showing the English units as on the Garmin GPS.

Domain knowledge: 1 inch is 2.54 centimeters. 1 foot is 12 inches. 1 mile is 5280 feet.