CS151.01 2009F *Functional Problem Solving*

Class 06: Computing with Symbols and Numbers

Held: Monday, 7 September 2009

Summary: We explore a bit more about data in Scheme, particularly the ways in which our version of Scheme supports numbers.

Related Pages:

- EBoard.
- Lab: Numeric Computation.
- Reading: Numeric Values & Symbolic Values.

Notes:

- EC: Women's Soccer vs. Central, 5:30 p.m., Wednesday.
- I've added instructions for lab writeups to the handouts area.
- Don't forget to fill out the RISC survey at `http://www.grinnell.edu/academic/psychology/faculty/dl/risc/`. It should take about fifteen minutes.
- The quiz is graded and returned. I do not report average grades or other statistics on grading.
- There was a bit of confusion on 2c on the quiz. Since `name` is a variable and undefined, the interpreter will respond with an error when it is used.
- A number of you neglected to include the quotation marks in string output. These marks tell you (as user of Scheme) that the result is a string, rather than some other type of value, such as a symbol.
- A question on the first writeup: Some folks wondered what me meant by "changing the size of the image". We want you to reflect what happens if we do the same drawing commands in a window of a different size.
- Are there other questions on the lab writeup or assignment?

Overview:

- Types.
- Kinds of Numbers.
- Modulo.
- Lab.

Types

- As you may have noted in your first experiments with Scheme, Scheme assigns *types* to variables.
- For example, a value might be a number, or a string, or an image identifier, or a procedure, or a drawing, or ....
- Computer scientists often think of types in two different ways:
    - *Data-driven*: A type is a set of values.

    - *Purpose-driven*: A type provides information on the valid operations that may be applied to a piece of data.
- We will alternate between the two definitions.
- Many languages (particularly the ones you've reported being familiar with) require you to assign a type to a variable when you declare that variable.
- Scheme does not require you to assign types ot variables; it checks the type of each operand when it executes a procedure.
    - Scheme also provides procedures that let you determine the type of a value.
- As the semester progresses, you will learn new types.
- As you learn each type, you'll learn a variety of things (that correspond, in some sense, to those two approaches):
    - How to express values in the type. For example, we express string values by surrounding them with double-quotation-marks and we express numbers in much the way we always have.
    - What operations are possible on values in the type. For example, we can use the addition operation (`+`) on numbers and we can use the `string-append` operation on strings.

Scheme's Numeric Types

- Instead of a general "numbers" type, Scheme provides a variety of kinds of numbers.
- Integers are numbers without a fractional component.
- Rational numbers can be expressed as the ratio of two integers.
- Real numbers appear somewhere on the number line.
    - In mathematics, real numbers can be rational or irrational.
    - In Scheme, real numbers are all rational.
- Complex numbers may include an *imaginary* component.
- You can (almost) always use an integer when a real is expected, but you cannot always use
- Scheme also represents some numbers exactly and some numbers inexactly. (That is, it approximates some numbers.)
    - It certainly has to approximate irrational numbers.
    - But it also approximates many other numbers.
    - It may surprise you to see which numbers are represented inexactly. (We'll return to this issue later.)
- Some important numeric predicates (procedures that return true or false): `number?`, `real?`, `integer?`, `exact?`, and `inexact?`.

Modulo

- The `mod` (modulo, modulus) operation is one of the trickier operations we use in this class (and we use it a lot).
- Essentially, `mod` is used to break up the number line into even chunks.
    - If you mod by 7, you break the number line up into chunks of size 7.
    - If you mod by 23, you break the number line up into chunks of size 23.
- For each chunk, we start counting at 0.
- For example

```
Number line:  -9 -8|-7 -6 -5 -4 -3 -2 -1| 0  1  2  3  4  5  6| 7  8  9 10 11
Modulo 7:      5  6| 0  1  2  3  4  5  6| 0  1  2  3  4  5  6| 0  1  2  3  4
```

- The `(modulo i n)` operation allows us to *cycle* through the numbers between 0 and *n*-1.

Lab

- Do the lab on numeric values.
- Be prepared to reflect.
