CS151.01 2009F *Functional Problem Solving*

Class 04: An Introduction to Scheme

Held: Wednesday, 2 September 2009

Summary: Today we begin your exploration of the Scheme programming language and the environment in which you will be devleoping algorithms in Scheme.

Related Pages:

- EBoard.
- Lab: Starting Scheme.
- Reading: The MediaScript Program-Development Environment & Beginning Scheme & How Scheme Evaluates Expressions (version 1).

Notes:

- Extra credit for attending any of the Rosenfield Human Rights Symposium talks over the next few days. (Jane Mayer at 4:15 today; Michael Grodin at 8:00 tonight; Lynn Hunt in the convo slot tomorrow.)
- My notes on assignment 1 (including my answers to all of your strange questions) are now available.
- I will go over the key concepts of today's readings and then give you a chance to ask questions before you start lab.
- Reading for this Friday: Programming the GIMP Tools.
- Syllabus change: We had a homework scheduled for this Friday. Now we don't.
- Assignment 2 is now available. It is due next Wednesday.

Overview:

- Why use programming languages?
- Scheme basics.
- Scheme history.
- Lab.

The Need for Formality

- Computer scientists often express their algorithms in *programming languages*, languages designed by other computer scientists to express algorithms.
  - Most programming languages can also be "understood" by computers.
  - More precisely, we can use a program to automatically translate programs written in these languages into instructions the computer can follows.
- In this class, you will express your algorithms in the programming language Scheme.
- In Scheme, as in most languages, you will explore the *syntax* (the structure of the things we write) and the *semantics* (what meaning we assign to the things we write).
- Some people wonder why we need artificial languages like Scheme, Pascal, C, Java, and the ilk,

given that we should be able to write algorithms in English.
    - In part, it's probably because the computer elite want to maintain their sense of superiority over the masses.
    - In greater part, it's because English and other "natural" languages can be ambiguous. At the very least, they have many similar structures that are interpreted very differently. Consider the classic pair of phrases.
    - Time flies like an arrow.
    - Fruit flies like a banana.
- Remember: *Computers are sentient and malicious.*
    - From the programmer's perspective, it often seems that they'll do their best to misinterpret whatever it is you write.

Scheme Basics

- Scheme is generally used in an *interactive* environment: You type syntactically valid expressions and Scheme returns their values.
    - Later this semester, we may explore how to use Scheme in different environments.
- Scheme's syntax is fairly simple: Since almost everything in Scheme involves the application of a function (a.k.a. operation, a.k.a. procedure) to some arguments (a.k.a. parameters), you write:
    - an open parenthesis;
    - the name of the function;
    - a space;
    - the arguments to the function (separated by spaces); and
    - a close parenthesis.
- For example,

```
(+ 2 3 4)
(sqrt 4)
(expt 2 3)
```

- If you remember this basic syntactic structure, you'll rarely get syntactic errors from Scheme.
- And if you believe that claim, I have a bridge to sell you.

Some Procedures You Should Know

- `+`: Sum its parameters.
- `*`: Compute the product of its parameters.
- `-`: Subtract the second parameter from the first.
- `(sqrt val)`: Compute the square root of *val*.
- `(expt v p)`: Compute v^p.
- `(abs v)`: Compute the absolute value of *v*.
- `(max v1 v2 ... vn)`: Determine the largest of the given values.
- `(min v1 v2 ... vn)`: Determine the smallest of the given values.

A Short History of Scheme

- Scheme is a variant of Lisp (the List processing language).
- Lisp is one of the oldest high-level programming languages in active use
  - It's from the late 1950's.
  - That is, it's from about the time of the invention of Cobol and only a few years after the invention of Fortran.
  - It predates C, C++, C#, Basic, and almost every other language used.
- John McCarthy (of MIT and then Stanford) designed Lisp to provide a language for programming in artificial intelligence.
  - At the time, many people believed that intelligence was grounded in symbolic processing.
- Lisp added many things to languages of the day:
  - Symbolic values
  - Dynamic lists as built-in data structures
  - Automatic memory management for those built-in data structures
  - Functions as values (we'll return to this in a few weeks)
- In later reflection, McCarthy indicated that some of these things were just luck. For example, the `lambda` that you'll learn about soon was just a "hmmm ... that sounds interesting, I'll put it in" fluke.
- Scheme was designed in the 1970's as a variant of Lisp more appropriate for teaching computer science (and doing many other things; it's more elegant).
  - Scheme has a slightly clearer semantics than Lisp (and a formal semantics, which is important).
  - Scheme adds some nifty features for advanced programmers.
  - But, at least for 151, you're doing Lisp-like programming.
- A variant of Scheme called Script-Fu was added to the Gimp in its early development, so it is tightly integrated with the Gimp.
- We've built a friendlier version of Script-Fu.

Lab

- Now that you have learned DrScheme and some background, it's time for the lab.
- I hope to spend the first few minutes at the beginning of the next class giving you a chance to reflect on your first real experiences with Scheme.
