

Class 04: Beginning Scheme

Held: Wednesday, 30 August 2006

Summary: Today we begin your exploration of the Scheme programming language (and of algorithm development) in earnest.

Related Pages:

- EBoard.
- Lab: Beginning Scheme.
- Reading: Beginning Scheme.

Notes:

- Testing.

Overview:

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

The Need for Formality

- In this class, you will express your algorithms in Scheme.
- Some people wonder why we need computer languages like Scheme, Pascal, C, Java, and the ilk.
- 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
 - Time flies like an arrow.
 - Fruit flies like a banana.
- Remember: Computers are sentient and malicious. 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'll see how to use Scheme in different environments.

- Scheme's syntax is fairly simple: Since almost everything in Scheme involves the application of a function to some arguments, you write:
 - an open parenthesis;
 - the name of the procedure;
 - a space;
 - the parameters to the procedure (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.

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 about the time of the invention of Cobol and only a few years after the invention of Fortran.
- 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 early 1980's (or maybe late 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.

Lab

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

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.