

## Beginning Scheme

**Summary:** We consider the purpose of Scheme and the structure of expressions in Scheme.

### Contents:

- Algorithms and Programs
- Procedure Calls
- Scheme Syntax
- Definitions
- More Procedures

## Algorithms and Programs

Our main objectives in this course are to learn about *algorithms*, step-by-step methods for solving problems, and to learn how to direct computers to perform such algorithms for us. A *programming language*, such as Scheme, is a formal notation in which one can express algorithms so exactly that a computer can perform them without any other assistance from human beings. The expression of an algorithm in such a notation is called a *program*, and the computer is said to be *executing* the program when it is performing the algorithm as directed.

Although not all of the problems that we'd like computers to solve are arithmetical, the simplest examples belong to that category, and we'll start with a few of them. Here, for instance, is a program, written in Scheme, that directs the computer to find the answer to the question "What is the square root of 137641?"

```
(sqrt 137641)
```

In order to make the Scheme environment answer that question, you need to learn how to work with Scheme. There are many different implementations of Scheme available. We'll use DrScheme, which you began to learn about in a previous lab. In DrScheme, you can work interactively with Scheme, typing a bit of code, finding a result, typing another bit of code, finding a result, and so on and so forth.

```
> (sqrt 137641)  
371
```

## Procedure Calls

The full Scheme language that DrScheme supports contains several hundred *primitive procedures* -- operations, such as finding the square root of a number, for which DrScheme can use prepackaged algorithms. Some programmers who are experts on square roots and on the idiosyncracies of our computers have figured out and written up a step-by-step method for computing the square root of any number, using only the very elementary transformations that the processor can perform. DrScheme recognizes `sqrt` as the name of this algorithm and knows where the processor instructions that carry it out are stored. When DrScheme receives a command to compute a square root, it recovers these

instructions and arranges for the processor to follow them.

A *procedure call* is a command that directs DrScheme to activate a procedure such as `sqrt`. (Note: `sqrt` is the name of the procedure, and `(sqrt 137641)` is the procedure call.) In Scheme, every procedure call begins with a left parenthesis and ends with the matching right parenthesis. Within the parentheses, one always begins by identifying the procedure to be called and then continues by identifying the *arguments*, the values that the procedure is supposed to operate on. The `sqrt` procedure takes only one argument, the number of which you want the square root, but other procedures take two or more arguments, and some need no arguments at all.

All arithmetic in Scheme is done with procedure calls. The primitive procedure `+` adds numbers together, the primitive procedure `-` subtracts one number from another. Similarly, the primitive procedure `*` performs multiplication, and the primitive procedure `/` performs division. The fact that in a procedure call the procedure is identified first makes calls to these procedures look different from ordinary arithmetic expressions: For instance, to tell DrScheme to subtract 68343 from 81722, one gives the command `(- 81722 68343)`.

Other Scheme procedures include `abs`, and `expt`. The Scheme procedure `abs` computes the absolute value of its argument. The Scheme procedure for raising a number to some power is `expt`.

## Scheme Syntax

As you may have noted, the appropriate way to write a Scheme expression is

```
(operation operand1 ... operandn)
```

That is, you parenthesize the expression (and any nontrivial subexpressions) and you place the operation before the operands.

It is harmless, though unproductive, to try to give DrScheme ordinary arithmetic expressions, in which the procedure is written between the operands.

## Definitions

DrScheme can also learn new names for things by reading definitions. Here's what a definition looks like:

```
(define days-in-a-week 7)
```

Like a procedure call, a definition begins and ends with matching parentheses. To distinguish between definitions and procedure calls, DrScheme looks at what comes immediately after the left parenthesis. In a definition, the keyword `define` must appear at that point. `Define` is *not* the name of a procedure; it is part of the syntactic structure of the Scheme programming language. Its only role is to serve as the mark of a definition.

After the keyword `define`, a definition contains the name being defined and an expression that identifies the value that the name should stand for. In this example, the name is `days-in-a-week`. (Notice that in Scheme a name can, and often does, contain hyphens internally.) The value that it names is the number 7.

Once DrScheme has seen this definition, it remembers that `days-in-a-week` stands for 7.

The value that gets a new name need not be a number; it can be anything, even a procedure. For example, if you don't like the name `*` for the multiplication procedure and would rather call it by the name `multiply`, just start each sequence of interactions with DrScheme by giving it the definition (`define multiply *`). (Alternately, place the definition in a file, and load the file in your interactions window.)

## More Procedures

At this point, I hope you're wondering what other useful and interesting procedures are built into Scheme. Section 6.2.5 of the *Revised<sup>5</sup> report on the algorithmic language Scheme* contains a list of the ones that are mainly about numbers, and that's only one section of the full roster of standard Scheme procedures. Fortunately, most of the primitive procedures perform small, simple jobs and are easily learned.

---

Copyright © 2007 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.