

Symbolic Values in Scheme

Summary: In this reading, we consider one of Scheme's central kinds of values, symbols.

Contents:

- Introduction
- Writing Symbols
- Operations on Symbols
- Symbols Versus Names
- Side Note: Quoting Other Values

Introduction

While your initial exercises in Scheme have been numeric, Scheme is not limited to numerical computation, but can also operate on pure symbols.

Scheme's ancestor, Lisp, was originally developed to aid in experiments in artificial intelligence. At the time, a leading theory suggested that intelligence emphasizes symbolic manipulation. Hence, it is sensible that Lisp and Scheme include symbols as a basic type. Evidence also shows that many programs most appropriately work on abstract symbolic value.

So, what is a symbol? A symbol is simply a word (usually) that we use to denote only itself. Unlike a variable, it has no associated value. Symbols are also *atomic*, we cannot split them apart (as we might a sequence of letters). The primary operation we perform on symbols is comparison (determining whether two symbols are the same).

Writing Symbols

When we want to refer to something as a value involved in a computation, rather than as the name of some other value, we put an apostrophe (usually pronounced "quote") in front of it. In effect, by quoting the symbol, we're telling Scheme to take it literally and without further interpretation or evaluation:

```
> 'sample  
sample
```

We can also create symbols using the `quote` operation.

```
> (quote sample)  
sample
```

Operations on Symbols

So, what can you do with symbols? Not a whole lot. You can determine if a value is a symbol using the `symbol?` procedure and you can determine if two symbols are the same using the `eq?` procedure. Note that Scheme uses `#t` for “yes” on `#f` for “no”. We’ll return to those value when we begin to consider conditionals.

```
> (symbol? 'sample)
#t
> (symbol? 23)
#f
> (eq? 'sample 'elpmas)
#f
> (eq? 'sample 'sample)
#t
```

Symbols Versus Names

Note that `'sample` (with the quote) is very different from `sample` (without the quote). In the first case, Scheme interprets it as a symbol (an atomic value). In the second, Scheme interprets it as a name for another value (e.g., something defined with `define`). At first, you may find the distinction a bit confusing. However, as you get used to programming in Scheme, the distinction will become natural.

```
> (define sample 85)
> 'sample
sample
> sample
85
> (symbol? 'sample)
#t
> (symbol? sample)
#f
> (eq? sample 'sample)
#f
```

The problem becomes even worse when you use a name as a symbol, and the name has not been defined.

```
> (symbol? 'elpmas)
#t
> (symbol? elpmas)
reference to undefined identifier: elpmas
```

Side Note: Quoting Other Values

Note that quote does not create a symbol. Rather, it tells Scheme to take something literally. Hence, you can quote many different things. For example, you can even quote Scheme expressions.

```
> '(+ 2 3)
(+ 2 3)
> (quote (* 2 3))
(* 2 3)
```

Although you can use quote in a variety of ways, I prefer that you limit your use to quoting symbolic values, at least for the first few weeks of class. My experience shows that those who quote other kinds of values early in the course careers end up with confusing results later in the course.

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.