Scheme Quick Reference
Coen 166/266

Using The Guile Interpreter

```
$ ssh myname@linux.dc.engr.scu.edu
[myname@linux60820 ~]$ guile
guile>
```

Add the following lines to your `~/.guile` file to enable the command history feature:
```
(use-modules (ice-9 readline))
(activate-readline)
```

To exit the interpreter, press <CTRL-D>

Data Types

String	"This is a string"
Character	#\c
List	(a b c)
Boolean	#f #t (anything other than #f is considered true)

Identifiers

Can be comprised of [A-Za-z0-9?!.+-*/<=>:$%^&_~@]
  ...except that they generally cannot begin with [0-9+-.]
    ...except that these are valid identifiers: + - ...
      ... and identifiers can begin with the two-character sequence ->
Identifiers are case-sensitive

Built-in Procedures

Evaluation
quote	prevent evaluation of an argument

List Manipulation
car	retrieve the head of a list
cdr	retrieve a list with all except the head
cons	add a value to the front of a list
list	create a list with the specified elements
append	concatenates multiple lists

Mathematical Operations
```
* / + -
```

Logical Operations
not and or	perform boolean comparisons
< > <= >= =	perform arithmetic comparisons
null?	determine if a list is null
equal?	compare two values for structural equivalence

Input/Output Operations
display	display a value to the terminal
newline	display a newline to the terminal
read	read a value from the terminal

Built-in Procedures (continued)

Conditional Operations

if
Syntax: `(if condition expr1 expr2)`
Behavior: If `condition` evaluates to anything other than `#f`, evaluate and return `expr1`. Otherwise, evaluate and return `expr2`.
Example:
```
guile> (if (and #f #t) "You win!" "Sorry!")
"Sorry!"
```

cond
Syntax: `(cond (cond1 expr1) (cond2 expr2) ... (condN exprN))`
Behavior: Return the expression associated with the first true condition.
Example:
```
guile> (cond ((> 12 13) "Hello")
             ((< 5 0) "There")
             ((= (+ 3 3) 6) "Waldo")
             (#t "Help!"))
"Waldo"
```

Complex Operations

define
Syntax: `(define symbol expr)`
Behavior: Specify a value for a symbol
Example (for defining a procedure):
```
guile> (define (square arg) (* arg arg))
guile> (square 3)
9
```

begin
Syntax: `(begin expr1 expr2 ... exprN)`
Behavior: Evaluate each expression and return the value of the last
Example:
```
guile> (begin (+ 2 2) (* 3 3) (- 16 2))
14
```

let
Syntax: `(let ((sym1 val1) (sym2 val2) ... (symN valN)) expr)`
Behavior: Evaluate `expr`, replacing each reference to `sym1` with `val1`, each reference to `sym2` with `val2`, etc.
Example:
```
guile> (let ((x 5)) (* x x))
25
```

Online Reference
The Scheme Programming Language, 4th Edition
```
http://www.scheme.com/tspl4/
```

Guile reference (includes a nice Scheme overview, too):
```
http://www.gnu.org/software/guile/manual/guile.pdf
```