Scheme Tutorial

for B551 (and B351)

Please download
Petite Chez Scheme
(Scheme.com)
If you haven't already...

Todd Holloway
Indiana University
08-31-2007

Part 1: Getting Started

What to expect…

- Lots of parenthesis ((((((((((()))))))))))
  - When debugging, check for mismatched parenthesis
- Lots of lists
  - Schemers love list processing
- Lots of recursion
  - Loops are often recursively defined
- And lots of lambdas
  - Pass variables, functions in many (often imaginative) ways

Using the Command-Line Interpreter…

Petite Chez Scheme Version 7.3

;; semi-colons are used for comments

- (+ 3 5) ; form is function followed by args
  - 8
- (+ (* 2 2) (* 5 5)) ; composition of functions
  - 29
- (length '(1 3 5 9 11)) ; first look at lists
  - 5

More basics…

- (eqv? 42 42) ; take a look at eq?, =, and equal? as well
  - #t
- (min 1 3 4 2 3) ; no limit on number of arguments
  - **1**
- (display 8)
  - **8**
- (let ((x 3) (y 4) (z 5)) (display (+ x 1))) ; local vars
  - 4

Note: This can be implemented using lambdas instead

Lists…

- (cons 1 '()) ; concatenation with an empty list
  - (1)
- (cons 1 (cons 2 (cons 3 (cons 4 '()))))
  - (1 2 3 4)

- (define x (list 1 2 3)) ; use define to a bind an object to a variable

- (car x) ; car returns the first item in a list
  - 1
- (cdr x) ; cdr is the list without the first element
  - (2 3)
- (cadr x) ; cddr, caddr, etc.
  - 2

Lambda…

- Syntax: (lambda formal-parameters body)

- ((lambda (x) (+ x 2)) 5)
  - 7

- Why?
  - Lambda (x) is asking for 1 argument
  - We pass in 5

More lambda…

```
;; pass in an unspecified number of args
(define sum
    (lambda (x . y)   ; the dot precedes a list
       (apply + x y))) ;(apply proc arg1 … args)


(sum 2 2 2 2)
       8
```

Part 2: A Little Deeper…

A Simple Recursive Program

```
(define factorial
   (lambda (n)
         (if              ; Syntax:  (if condition consequence alt)
          (= n 0)
         1
         (* n (factorial (- n 1))))))

> (factorial 6)
720
```

More programs…

```
;; Mutual recursion
(define is-even?
   (lambda (n)
         (if (= n 0) #t (is-odd? (- n 1)))))

 (define is-odd?
   (lambda (n)
         (if (= n 0) #f (is-even? (- n 1)))))

> (is-odd? 13)
#t
```

On Side Effects

- C coders, in particular, often try to write Scheme in a procedural fashion

```
(define x 3)
(define ugly
   (begin
         (set! x 4) ; global change
         (set! x (+ 1 x))))

> ugly
> x
5
```

Programming tips…

- Source files usually end in .ss or .scm

> (cd "C:\\B551")

> (load “filename.ss")

- Use EMACS!!
  - Xemacs.org
- Or Eclipse
  - Eclipse.org
  - schemeway.sourceforge.net (scheme plugin)

Part 3: A Little Linear

Matrices

- We will use matrix algebra in our first homework assignment to implement a neural net (B551)

- Use the matrix functions from Scheme and Art of Programming chapter 9.4, available at http://ftp.cs.indiana.edu/pub/eopl/sap-source.ss
  - Copy the vector-generator (9.21) and matrix (9.3 – 9.39) procedures into the interpreter or place them in a file called matrix.ss, and place (load “matrix.ss”) at the top of your assignment file.

Matrices

- Challenge: What does this do?

```
(define matrix-generator
   (lambda (gen-proc)
         (lambda (nrows ncols)
          (let ((size (* nrows ncols)))
           (let ((vec-gen-proc
                  (lambda (k)
                    (if (< k size)
                        (gen-proc (quotient k ncols) (remainder k
                        ncols)) ncols))))
             ((vector-generator vec-gen-proc) (add1 size)))))))
```

Matrices

```
(define make-zero-matrix
        (matrix-generator (lambda (i j) 0)))

> (make-zero-matrix 3 5) ; 3 rows, 5 columns
#16(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5)
```

Homework 0.5

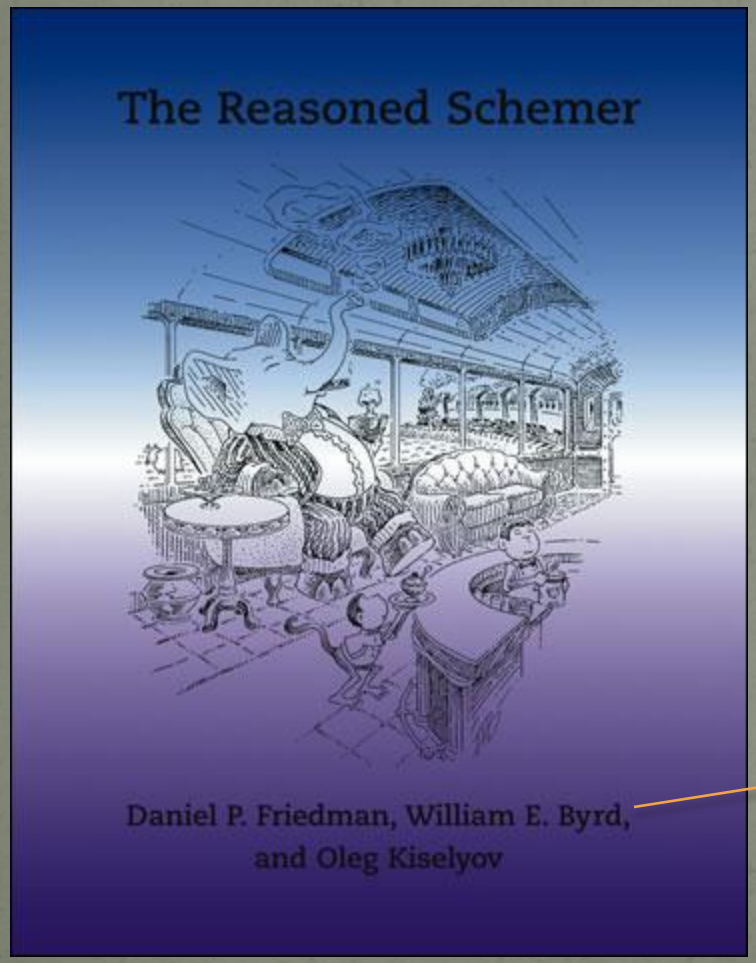
- Notes:
  - Not graded, but useful for doing Homework 1—NN
  - Use the code from chapter 9.4 of Scheme and the AOP

1. Create a 20 x 10 matrix of zeroes
2. Change the first column into ones
3. Transpose the matrix
4. Subtract 20 from [0,15]

5. (Additional Practice) Create a procedure *replace* which takes a list of words, a word to replace, and a replacement word, and returns a list with the replacements made.

Part 4: Scheme Resources

Scheme resources…



Will Byrd is taking B551 ☺

Scheme Resources

- Scheme.com/resources.html
  - Including the language manual
- Schemers.org
  - Comprehensive list of resources

- How to Debug Scheme
  - www.cs.indiana.edu/chezscheme/debug/

- Planet Scheme (scheme.dk/planet/)
  - Blog aggregator
- Lambda-the-ultimate.org
  - Group blog
- Norvig.com
  - Google's Director of Research

- **Any others?**