MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.090—Building Programming Experience
IAP 2005

Lecture 2

Scheme

1. **Basic Elements**

   (a) *self-evaluating* - expressions whose value is the same as the expression.

   (b) *names* - Name is looked up in the symbol table to find the value associated with it. Names may be made of any collection of characters that doesn't start with a number.

2. **Combination**
   `(` *procedure arguments-separated-by-spaces* `)`
   Value is determined by evaluating the expression for the procedure and applying the resulting value to the value of the arguments.

3. **Special Forms**

   (a) *define* - `(define` *name value*`)`
   The name is bound to the result of evaluating the the value. Return value is *unspecified.*

   (b) *if* - `(if` *test consequent alternative*`)`
   If the value of the test is not false (#f), evaluate the consequent, otherwise evaluate the alternative.

   (c) *lambda* - `(lambda` *parameters body*`)`
   Creates a procedure with the given parameters and body. Parameters is a list of names of variables. Body is one or more scheme expressions. When the procedure is applied, the body expressions are evaluated in order and the value of the last one is returned.

Problems

1. *Evaluation* - For each expression:

   (a) Write the type of the expression

   (b) Write your guess as to the expression's return value. If the expression is erroneous simply indicate "error" for the value. If the expression returns an unspecified value, write whatever you want! If the expression returns a procedure, indicate "procedure" for the value.

   (c) Evaluate the expression, and copy the response from the *scheme* buffer.

```
(lambda (x) x)

((lambda (x) x) 17)

((lambda (x y) x) 42 17)

((lambda (x y) y) (/ 1 0) 3)

((lambda (x y) (x y 3)) (lambda (a b) (+ a b)) 14)
```

2. *Writing Procedures* - Write the procedure indicated. Then test it in scheme to make sure it works.

   (a) Write a procedure `cube` that returns the cube of it's input.

   ```
   (define cube
   ```

   (b) Write a procedure `the-answer?`, which returns true (`#t`) if the input is the number 42.

   ```
   (define the-answer?
   ```

   (c) Write a procedure `sign` that returns 1 if it's input is positive, -1 if it's input is negative, and 0 if it's input is 0.

   ```
   (define sign
   ```

(d) Given a margin width `m`, which is both the top, bottom, left, and right margin of the page, write a procedure that computes the "usable" (non margin) area of the 8.5in by 11in sheet of paper.

```
(usable-page 0)
;Value: 93.5
(usable-page 1)
;Value: 58.5

(define usable-page
```

(e) Write a procedure that when given a width, returns the length of the most beautiful rectangle having that width. According to studies, the most beautiful rectangle is one whose ratio of length to width is the golden ratio. The golden ratio can be most easily be expressed as (sqrt(5)+1)/2.

```
(beautiful-rectangle 1)
;Value: 1.618033988749895
(beautiful-rectangle 34.5)
;Value: 55.82217261187137

(define beautiful-rectangle
```

(f) Write a procedure that computes the positive root of the quadratic polynomial using the quadratic formula. The positive root is the larger of the two roots. If the polynomial has complex roots, your procedure should return the string "complex roots".

```
(postive-root 1 -2 1)
;Value: 1
(positive-root 3 1 3)
;Value: "complex roots"

(define postive-root
```

3. *Biggie-Sizing!*

   Suppose we're designing an point-of-sale and order-tracking system for Wendy's[1]. Luckily the Über-Qwuick drive through supports only 4 options: Classic Single Combo (hamburger with one patty), Classic Double With Cheese Combo (2 patties), and Classic Triple with Cheese Combo (3 patties), Avant-Garde Quadruple with Guacamole Combo (4 patties). We shall encode these combos as 1, 2, 3, and 4 respectively. Each meal can be *biggie-sized* to acquire a larger box of fries and drink. A *biggie-sized* combo is represented by 5, 6, 7, and 8 respectively.

   (a) Write a procedure named `biggie-size` which when given a regular combo returns a *biggie-sized* version.

   (b) Write a procedure named `unbiggie-size` which when given a *biggie-sized* combo returns a non-*biggie-sized* version.

   (c) Write a procedure named `biggie-size?` which when given a combo, returns true if the combo has been *biggie-sized* and false otherwise.

   (d) Write a procedure named `combo-price` which takes a combo and returns the price of the combo. Each patty costs $1.17, and a *biggie-sized* version costs $.50 extra overall.

[1] 6.090 and MIT do not endorse and are not affiliated with Wendy's in any way. They merely capitalize on the pleasant way "biggie-size" rolls off the tongue.

Recursion

More Problems

4. Write `expt`, a procedure that raises `x` to the `n`th power. You may assume that `n` is non-negative and integer.

```
(expt 2 2)
;Value: 4
(expt 2 3)
;Value: 8
```

```
Plan:
```

```
(define expt
  (lambda (x n)
```

5. Write `remainder`, a procedure that computes the remainder of `x` divided by `y`.

```
(remainder 2 5)
;Value: 2
(remainder 7 5)
;Value: 2
```

```
Plan:
```

```
(define remainder
  (lambda (x y)
```

6. Write `fib`, a procedure that computes the `n`th fibonacci number.

```
(fib 0)
;Value: 1
(fib 1)
;Value: 1
(fib 2)
;Value: 2
(fib 6)
;Value: 13
```

```
Plan:
```

```
(define fib
  (lambda (n)
```