

6.001 Tutorial 1 – Solutions

David Ziegler

September 13, 2004

1 General Information

Your TA: David Ziegler
Email: david@ziegler.ws
Cell: 617-429-6685
Tutorial webpage:
<http://david.ziegler.ws/6.001/>

2 6.001 Lab

- 34-501, 50 Vassar Street (for ordering food late at night)
- Outer door combination: 47519
- Inner door combination: 72962*
- Friendly lab assistants are available!

3 Due Dates

- Problem set 1 — due **tomorrow** at midnight! Don't wait until the last couple hours to turn it in, because the server gets slow.
- Lecture 3 problems — due **Wednesday** at 10 am.
- Lecture 4 problems — due **Friday** at 10 am.
- Project 0 — due **Wednesday** at 6 pm.

4 Scheme

- Why do we like Scheme?
- **Very** simple syntax – you can learn it in under an hour.
- Focus on learning *programming*, not *language*.
- It's actually used in the real world! Yahoo! Store, artificial intelligence, ...

5 Types of Expressions

- **Constants:**
42, "hello", 3.1415926535...
These are self evaluating – the value is the constant itself.
- **Names:**
a, -, -\$\$~foo
The value of a name is found by looking up the name in the table. Later on in the course, we'll explain how this really works.
- **Combinations:**
(procedure argument argument ...)
To find the value of a combination, first evaluate each subexpression (in any order). Then, apply the value of the procedure to the values of the arguments.
- **Special Forms:**
(define name value)
(if test consequent alternate)
(lambda (arg1 arg2 ...) body)
Each special form has a different rule for evaluation.

6 define

(define name value)
To evaluate a define expression, first evaluate value, then stick a new entry in the table, with name and the value of value. This *binds* the name to the value of value.

7 lambda

(lambda (arg1 arg2 ...) body)
The list of parameters can have any number of names – even zero. The body is a bunch of Scheme

expressions (but at least one). When the procedure is applied, each expression is evaluated, and the value of the last one is returned.

To evaluate a lambda, we create a procedure object and return a pointer to it, but *do not evaluate the arguments or the body*. The body is only evaluated when the procedure is applied later.

8 Syntactic Sugar

```
(define name
  (lambda (arg1 arg2 ...) body))
(define (name arg1 arg2 ...) body)
```

Since you often need to do the first form, Scheme provides *syntactic sugar* for this pattern. The two are *identical*.

9 if

```
(if test consequent alternate)
```

To evaluate an if expression, evaluate the test. If the value is *not* #f, the value of the entire expression is the value of the consequent. Otherwise, the value is the value of the alternate.

Why does if need to be a special form?

10 Problems!

```
;; This procedure should return the
;; larger of the two quadratic roots
;; of the quadratic ax^2+bx+c
(define quadratic-root
  (lambda (a b c)
    (/ (+ (- b) (sqrt (- (* b b)
                        (* 4 a c))))
       (* 2 a))))
```

```
;; This procedure should return the
;; remainder of x divided by y
(define remainder
  (lambda (x y)
    (if (< x y)
        x
        (remainder (- x y) y))))
```

```
;; This procedure should return #t
;; if x is divisible by y, and #f
;; otherwise
(define divisible?
  (lambda (x y)
    (= (remainder x y) 0)))

;; This procedure should return the
;; nth fibonacci number
;; (fibonacci 0) => 0
;; (fibonacci 1) => 1
(define (fibonacci n)
  (if (< n 2)
      n
      (+ (fibonacci (- n 1))
         (fibonacci (- n 2)))))

;; This procedure should return n
;; factorial
;; 3! = 6
;; 5! = 120
(define (fact n)
  (if (< n 2)
      1
      (* n (fact (- n 1)))))
```