MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Spring Semester, 1999

**Recitation – Wednesday, February 3**

# 0. Announcements

- **Section Staff:** RI: Mike Leventon    `leventon@ai.mit.edu`
                      TA: Sandia Ren       `sren@mit.edu`
                      TA: Hooman Vassef  `hvassef@mit.edu`     (Half of section 12)
- **Collaboration Policy:** Read carefully in the handout.
- **Attendance:** Lecture recommended. Section and Tutorials required.
- **Problem Sets:** No late problem sets accepted. Psets are an important part of your grade. Start them early!
- **Course Web Page:** `http://mit.edu/6.001`
- **Section Web Page:** `http://www.ai.mit.edu/people/leventon/6001`
  Section notes (with solutions) will be posted here.

# 1. Rules for Scheme

**To evaluate an expression**, follow these rules:

- A **numeral** evaluates to the **number**
- A **name** evaluates to **the value associated with that name**
- A **lambda expression** evaluates to a **procedure object**
- A **combination** is evaualted as follows:
  1. **Evalute** the *subexpressions in any order*
  2. **Apply** the *value of the operator subexpression* to the *values of the remaining subexpressions*.

**To apply a procedure to its arguments**, evaluate the body of the procedure where each parameter is replaced by the corresponding value.

**To evaluate a define**, evaluate the body of the define, and associate that value with the name listed in the define.

# 2. Simple Examples

To what do the following expressions evaluate (assume they are evaluated in sequence)?

```
⇒ 7
 ;Value: 7
⇒ -
 ;Value: #[compiled-procedure ..]
⇒ (+ 2 4)
 ;Value: 6
⇒ (* (- 5 3) (/ 9 3))
 ;Value: 6
⇒ (7 - 4)
 ;Error: The object 7 is not applicable
```

## 3. More Examples

To what do the following expressions evaluate (assume they are evaluated in sequence)?

```
⇒ (lambda (x) (* x x))                    ⇒ (define cube (lambda (x) (*x x x)))
 ;Value: #[compound-procedure ...]         ;Value: "cube --> #[compound-procedure ...]
⇒ ((lambda (x) (* x x)) 5)                ⇒ (cube 3)
 ;Value: 25                                ;Error: "Unbound variable: *x"
⇒ (define double (lambda (x) (* 2 x)))    ⇒ (define + 3)
 ;Value: "double --> #[compound-procedure ...]  ;Value: "+ --> 3"
⇒ (double (double 6))                     ⇒ (define - 6)
 ;Value: 24                                ;Value: "- --> 6"
⇒ (double double)                         ⇒ (* + -)
 ;Error: #[compound-procedure] passed to multiply  ;Value: 18
```

## 4. Writing a Procedure

Define a procedure called `average` that computes the average of its two numeric arguments.

```
(define average
  (lambda (x y)
    (/ (+ x y) 2)))
```

## 5. Substitution Model

Use the substitution model to evaluate the following expression: `(average 4 (double 4))`

```
⇒ ([procedure] 4 (double 4))
⇒                ⇒ (double 4)
⇒                ⇒ ([procedure] 4)
⇒                ⇒ (* 2 4)
⇒ ([procedure] 4 8)
⇒ (/ (+ 4 8) 2)
⇒ ([procedure /] (+ 4 8) 2)
⇒                ⇒ (+ 4 8)
⇒ ([procedure /] 12 2)
⇒ 6
```

## 6. Subtleties

Consider the following two definitions below. How are they similar and how do they differ?

```
(define plus +)

(define add
  (lambda (x y)
    (+ x y)))
```

## 7. More Subtleties

What do think should be the values of the following expressions? (Note: these are **not** things you have to memorize.)

```
⇒ (+ 4)                               ⇒ (+)
 ;Value: 4                             ;Value: 0
⇒ (- 3)                               ⇒ (*)
 ;Value: -3                            ;Value: 1
⇒ (/ 5)                               ⇒ (-)
 ;Value: 0.2                           ;Error: The procedure #[compiled-procedure] requires
⇒ (/ 60 5 2 3)                        at least one argment
 ;Value: 2
```