Notes 1.34-1.43

- First class computational element
  - Functions are treated like any other variable
    - May be named by variables
    - May be passed as arguments to procedures
    - May be returned as the results of procedures
    - May be included in data structures
- A function with lambda and without lambda has NO DIFFERENCE

Lambda

- (lambda (<*formal-parameters*>) <*body*>)
  - (define (plus4 x) (+ x 4))
  - (define plus4 (lambda (x) (+ x 4)))
  - They are equivalent, except lambda has no name for the procedure

- (lambda (x) (+ x 4))
  the procedure of an argument x that adds x and 4

Let in Local Variables

- Lambda can create local variables
  - Instead of writing a supplementary procedure, use lambda
- Special form: let
  - Instead of using lambda to define a procedure
  - (let ((<var_1> <exp_1>)
    (<var_2> <exp_2>)
    ⋮
    (<var_n> <exp_n>))
    <*body*>)

Half-interval

- If $f(a) < 0 < f(b)$, there must be at least one zero
  - Let x be the average of a & b and compute f(x)
  - If f(x) is greater than zero, it must be between a & x
  - If f(x) is less than zero, it must be between x & b
- Keep finding the midpoint until it is close enough to zero

Fixed point

- If a fixed point on the function satisfies f(x) = x
- Takes as inputs a function and an initial guess and produces an approximation to a fixed point of the function
    - Computer keeps guessing until it reaches a fixed point

** Taking a procedure/function as an input to another function

Procedures as returned value

- So far, procedures return a value
- However in scheme, procedures can return another procedure

```
; repeated will be on the test!
; where a function returns a function
```

A procedure that returns a procedure must use lambda!!