# 3. Functional Programming & Scheme

Lecture 03 -- Functional Programming.pdf

`(define square (lambda (x) (* x x))) -> ((define (square x) (* x x))`

The value of a lambda expression is a procedure

Heavy emphasis on the modularity of functional programming, capturing the common patters and so on.

- Formal parameter, argument
  n: formal parameter, 5: argument

## Substitution Model

A way to figure out what happens during evaluation, not really what happens in the computer

- Compound procedure: Evaluate the body of the procedure, with each parameter replaced (substituted) by the corresponding operand.
  `(average 5 (square 3))` -> `(average 5 (* 3 3))`
- Primitive Procedure: Just do it
  `(/ (+ 5 9) 2)` -> `(/ 14 2)`

Last part: How to write recursive algorithms, been taught since comp100 and comp106

## Iterative vs Recursive approaches

| Aspect | Iterative Approach | Recursive Approach |
| --- | --- | --- |
| Definition | Uses recursive calls WITHOUT pending operations. | Uses recursive calls WITH pending operations. |
| Memory Usage | Typically uses constant space since the call stack doesn't grow. | Uses more memory due to continious call stack growth |
| Readability | Can be more straightforward for simple tasks. | Can be more intuitive for problems that have a naturally recursive structure (e.g., tree traversal). |
| Efficiency | Might be more efficient for some problems due to reduced overhead. | Might have more overhead due to repeated function calls. Can be optimized using techniques like memoization. |