Procedural *vs* Functional Style

procedural = imperative

Abstracting Expressions

- Scale a 2D-point `p` by a factor `c`

```
(define  scale
  (lambda (c  p)
      (cons
             (* c  (first p))
             (list (* c (second p)))
      )
))
```

- Can abbreviate a fixed and finite number of function applications for readability, modifiability, and reusability.

Procedural/Imperative *vs* Functional

Procedural/Imperative	Functional
• *Program*: a sequence of instructions for a von Neumann m/c.	• *Program*: a collection of function definitions (m/c independent).
• Computation by instruction execution.	• Computation by term rewriting.
• *Repetition*: Iteration.	• *Repetition*: Recursion.
• Modifiable or updateable variables.	• *"Assign-only-once"* variables.

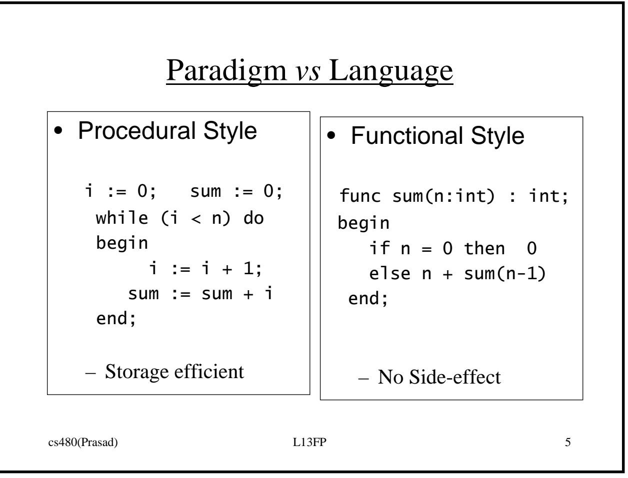
Functional Style : Illustration

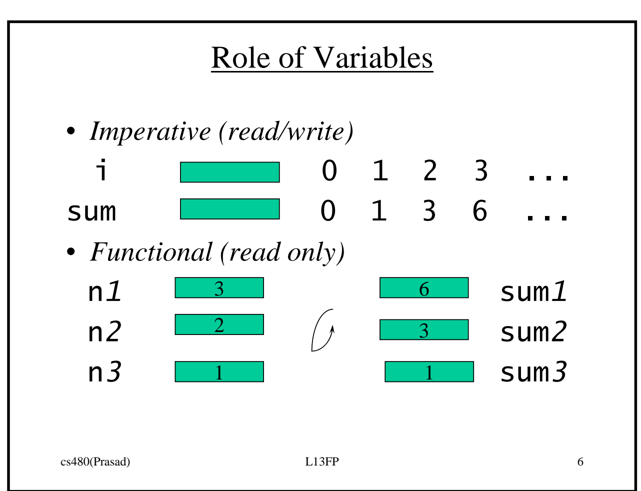
- Definition : *Equations*

```
sum(0)  = 0
sum(n)  =  n + sum(n-1)
```

- Computation : *Substitution* and *Replacement*

```
      sum(2)
=    2 + sum (2-1)
=    …
=    3
```





Bridging the Gap

- A ***tail recursive*** definition can be automatically optimized for space by translating it into an equivalent `while`-loop.

```
func sum(n : int, r : int) : int;
begin
      if  n = 0   then   r
      else   sum(n-1, n+r)
end
```

  - So, Scheme does not have loop-constructs.

Iteration *vs* Recursion

Recursion = Iteration + Stack

Iteration = Tail Recursion