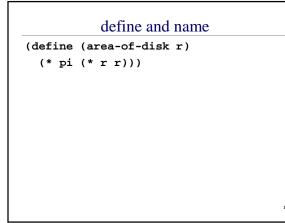
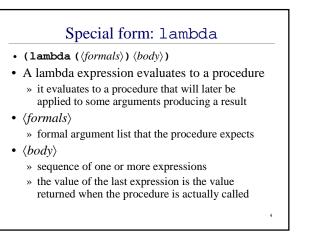
Topic #3: Lambda

CSE 413, Autumn 2007 Programming Languages

Scheme procedures are "first class"

- Procedures can be manipulated like the other data types in Scheme
 - » A variable can have a value that is a procedure
 - » A procedure value can be passed as an argument to another procedure
 - » A procedure value can be returned as the result of another procedure
 - » A procedure value can be included in a data structure





"Define and name" with lambda

```
(define area-of-disk
(lambda (r)
```

```
(* pi (* r r))))
```

"Define and use" with lambda

• ((lambda (r) (* pi r r)) 1)

Separating procedures from names

- We can treat procedures as regular data items, just like numbers
 - » and procedures are more powerful because they express behavior, not just state
- We can write procedures that operate on other procedures applicative programming

define min-fx-gx

(define (min-fx-gx f g x) (min (f x) (g x)))

apply min	-fx-gx
(define (identity x) x)	
(define (square x) (* x x))	
(define (cube x) (* x x x))	
<pre>(define (min-fx-gx f g x) (min (f x) (g x)))</pre>	
(min-fx-gx square cube 2)	; (min 4 8) => 4
(min-fx-gx square cube -2)	; (min 4 -8) => -8
(min-fx-gx identity cube 2)	; (min 2 8) => 2
(min-fx-gx identity cube (/ 1 2	<pre>(min 1/2 1/8) => 1/8</pre>

; define a procedure `s-fx-gx' that takes: ; s - a combining function that expects two numeric arguments ; and returns a single numeric value ; f, g - two functions that take a single numeric argument and ; return a single numeric value $f(x)$ or $g(x)$; x - the point at which to evaluate $f(x)$ and $g(x)$; s-fx-gx returns $s(f(x),g(x))$
(s-fx-gx min square cube 2); => $(min 4 8) = 4$
(s-fx-gx min square cube -2); => $(min 4 -8) = -8$
(s-fx-gx + square cube 2) ; => (+ 2 8) = 12
(s-fx-gx - cube square 3) ; => (- 27 9) = 18
10

