Higher-order Functions

+hof.rkt
Topics

- Functions are first-class.
- Using first-class/higher-order functions
- Map and filter

- Next time: getting the semantics right
First-class and higher order functions

Functions are first-class values, can be used or created wherever we use or create any other values:

– Arguments to (higher order) function calls
– Results of (higher order) function bodies
– Stored in cons cells or other data structures
– Bound (named) by variables
– …

Higher order functions take or return other functions.

Powerful ways to:

– factor out common functionality
– parameterize general patterns with specific behavior
Function closures support lexical scope for nested functions.

*Sneak peak:*

- Function bodies can use any bindings in scope where function is defined, *including from outside the function definition.*
- Distinct concept from *first-class functions*
- Back to this powerful idea soon!
Functions as arguments

(define (map-pair f pair)
  (cons (f (car pair)) (f (cdr pair)))))

Elegant strategy for factoring out code for common patterns of data manipulation.

Combines well with anonymous functions.

[See code examples in hof.rkt]
A style point

(if #t #f)

(lambda (x) (f x))

× (n-times (lambda (x) (cdr x)) 2 (list 1 2 3 4))

✓ (n-times cdr 2 (list 1 2 3 4))
Map

\[(\text{define } (\text{map } f \text{ elems})\]
\[(\text{if } (\text{null? } \text{elems})\]
null
\[(\text{cons } (f (\text{first } \text{elems}))\]
\[(\text{map } f (\text{rest } \text{elems}))\]
\)
(define (filter f elems)
  (if (null? elems)
      null
      (if (f (first elems))
          (cons (first elems)
                (filter f (rest elems)))
          (filter f (rest elems))))))
Rewrite the list practice functions

• Which functions could be built using map/filter?
• For which functions does this feel more or less elegant than your original implementation?
Generalizing

Our examples of first-class functions so far:
  – Take one function as an argument to another function
  – Process a number or a list

But first-class functions are useful anywhere for any kind of data
  – Pass several functions as arguments
  – Put functions in data structures (tuples, lists, etc.)
  – Return functions as results
  – Write higher-order functions that traverse other data structures

Powerful idioms to:
  – factor out and reuse common functionality
  – parameterize general patterns with specific behavior
  – clearly communicate high-level meaning/intent