Higher-order Functions

+hof.rkt
Topics

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

• Later: 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: `hof.rkt`

```
(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 `hof.rkt`
A style point

(if x #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))
(define (map f elems)
  (if (null? elems)
      null
      (cons (f (first elems))
            (map f (rest elems)))))

argument list
  v1 → v2 → ... → vn

result list
  (f v1) → (f v2) → ... → (f vn)
(define (filter f elems)
  (if (null? elems)
      null
      (if (f (first elems))
          (cons (first elems)
                (filter f (rest elems)))
          (filter f (rest elems))))))
List practice with HOFs: lists.rkt

• 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