Today's Lecture

- Compounds & variants are powerful features
  - But can they help us with the "fixed size input" problem?
- Building lists
- Taking them apart
- Taking them apart, sensibly
Lists: A Complete Definition

- All we need is two constructors:
  - `(define-struct empty ())`
  - `(define-struct cons (first rest))`

- Basically, the template will be:

```
(cond
  [(empty?) …]
  [else (…(first x)…(rest x)…)]
)```
Constructing a List

- Things like:
  - (define list0 empty)
  - (define list1 (cons 1 empty))
  - (define list2 (cons 2 (cons 1 empty)))
  - (define list3
      (cons 3 (cons 2 (cons 1 empty)))))

- Could this trick be useful?
- Is there anything fishy here?
Deconstructing a list

- \(\text{first}\ ((\text{cons}\ 2\ ((\text{cons}\ 1\ \text{empty}))))\) → 2
- \((\text{rest}\ ((\text{cons}\ 2\ ((\text{cons}\ 1\ \text{empty}))))))\) → \((\text{cons}\ 1\ \text{empty})\)
- \((\text{first}\ ((\text{rest}\ ((\text{cons}\ 2\ ((\text{cons}\ 1\ \text{empty}))))))))\) → \((\text{first}\ ((\text{cons}\ 1\ \text{empty})))\) → 1
- \((\text{rest}\ ((\text{rest}\ ((\text{cons}\ 2\ ((\text{cons}\ 1\ \text{empty}))))))\)\) → \((\text{rest}\ ((\text{cons}\ 1\ \text{empty})))\) → \text{empty}
What's the Contract for List?

- A list (of length n) is
  - empty, or ; base case
  - (cons x y) where
    - x is a number, and y is list (length (n-1))
- Have we seen this before?
- People often drop the "n"
  - What happens when we do that?
A More Natural Template

- Templates should correspond to contracts:

  (define (f x)
    (cond
      [(empty? x) …]
      [else (…) (first x) (f (rest x)) …)])
A Function on Lists

(define (f x)
  (cond
    [(empty? x) 0]
    [else (+ 1 (f (rest x)))]))

- Anything funny happen here?
- (f (cons 7 (cons 8 empty)))
  \[ \rightarrow (+ 1 (f (cons 8 empty))) \]
  \[ \rightarrow (+ 1 (+ 1 (f empty))) \]
  \[ \rightarrow (+ 1 (+ 1 0)) \]
  \[ \rightarrow 2 \]
Another Function on Lists

(define (f x)

(cond x

[(empty? x) 0]

[else (+ (first x) (f (rest x)))]])))

- Anything funny happen here?
- (f (cons 7 (cons 8 empty)))
  → (+ 7 (f (cons 8 empty)))
  → (+ 7 (+ 8 (f empty))) → (+ 7 (+ 8 0)) → 15