CS3L:
Introduction to Symbolic Programming

Lecture 24:
Review for lists, **`map`** and **`member`**

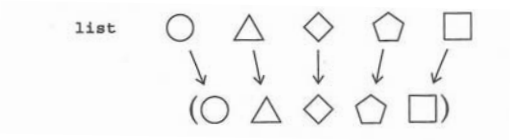
Summer 2008 Colleen Lewis
colleenL@berkeley.edu

Today

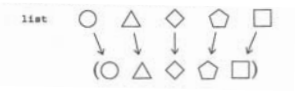
- Midterms grades are up! Mean = 22.9/36
- Review of List stuff
  - `list`
  - `cons`
  - `Append`
- `member versus member?`
- HOFs for Lists
  - `map`
  - `filter`
  - `reduce`

`list`

- Takes any number of arguments and puts them in a list



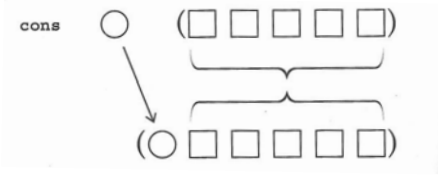
`list`



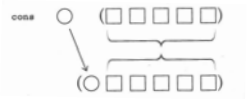
- Examples
  - `(list 'cat 'dog)` ➔ `'(cat dog)`
  - `(list '(cat) '(dog))` ➔ `'((cat) (dog))`
  - `(list '(cat) 'dog)` ➔ `'((cat) dog)`
  - `(list 'cat '(dog))` ➔ `'(cat (dog))`
  - `(list 'cat '())` ➔ `'(cat ())`
  - `(list '() 'dog)` ➔ `'(() dog)`
  - `(list '(cat ()) 'dog)` ➔ `'((cat ()) dog)`

`cons`

- Takes two arguments
- Makes the first arg the car of the new list
- Makes the second arg the cdr of the new list
- The second argument MUST be a list



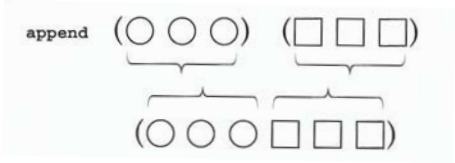
`cons`



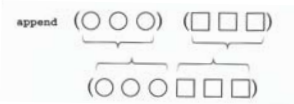
- Examples
  - `(cons 'cat '( dog ))` ➔ `'(cat dog)`
  - `(cons '(cat) '( dog ))` ➔ `'((cat) dog)`
  - `(cons 'cat '())` ➔ `'(cat)`
  - `(cons '() '( () dog ))` ➔ `'(() () dog)`
  - `(cons '(cat) 'dog` ➔ `'((cat) . dog)`

append

- Takes two lists and turns them into one
- Both arguments MUST be lists



append



- Examples
  - `(append '(cat) '(dog))` ➔ `'(cat dog)`
  - `(append '(cat) '())` ➔ `'(cat)`
  - `(append '() '(dog))` ➔ `'(dog)`
  - `(append '(cat) '(()))` ➔ `'(cat ())`
  - `(append '(()()) '(dog))` ➔ `'(() () dog)`

accessors

sentence & word stuff	list stuff
first	car
butfirst	cdr
last	☹
Butlast	☹

other procedures

sentence & word stuff	list stuff
empty?	null?
sentence?	list?
item	list-ref
sentence	list

other procedures

sentence & word stuff	list stuff
member?	member
every	map
keep	filter
accumulate	reduce

member? versus member

- For sentences
  - member? returns #t or #f
- For lists
  - member returns the rest of the list after that element or #f

member

(member 'a '(bbb a c)) ➔ (a c)
(member 'x '(bbb a c)) ➔ #f
(member '(mike clancy)
'((clint ryan) (mike clancy) (emily watt)))
➔((mike clancy) (emily watt))
(member 'clancy
'((clint ryan) (mike clancy) (emily watt)))
➔ #f

map

(map *procedure list1 list2…* **)**

- *procedure*
  - a procedure that takes in **some # of arguments**
- *Some # of sents*
  - The number of sentences MUST match the number of arguments that the procedure takes

map

```
(define (add-2-nums x y)
    (+ x y))

(map add-2-nums '(1  2  3)
                '(4  5  6))
                  ↓  ↓  ↓
              ➔'(5  7  9)
```

map

```
(define (add-3-nums x y z)
    (+ x y z))

(map add-3-nums '(1   2   3)
                '(4   5   6)
                '(7   8   9))
                  ↓   ↓   ↓
             ➔'(12  15  18)
```

map

```
(map cons  '(1    2    3)
           '((4)  (5)  (6)))
             ↓    ↓    ↓
       ➔'((1 4) (2 5) (3 6)
```