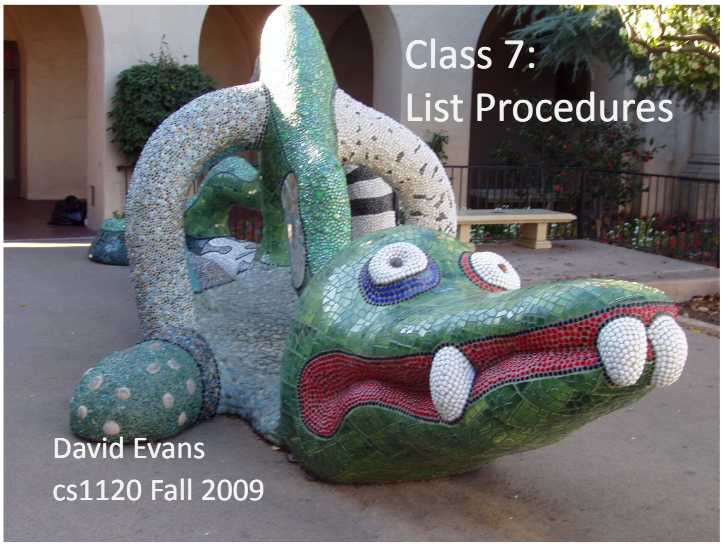

List Recap

- A *List* is either:
  (1) a Pair whose second part is a *List*
  or (2) **null**
- Pair primitives:

(cons a b)	Construct a pair <a, b>
(car pair)	First part of a pair
(cdr pair)	Second part of a pair

List Examples

```
> null
()
> (cons 1 null)
(1)
> (list? null)
#t
> (list? (cons 1 2))
#f
> (list? (cons 1 null))
#t
```


More List Examples

```
> (list? (cons 1 (cons 2 null)))
#t
> (car (cons 1 (cons 2 null)))
1
> (cdr (cons 1 (cons 2 null)))
(2)
```


List Procedures

- **Be *very* optimistic!** Since lists themselves are recursive data structures, most problems involving lists can be solved with recursive procedures.
- **Think of the simplest version of the problem**, something you can already solve. This is the base case. For lists, this is usually when the list is **null**.
- Consider how you would solve the problem using the result for a slightly smaller version of the problem. This is the recursive case. For lists, the smaller version of the problem is usually the **cdr** of the list.

list-trues

Define a procedure that takes as input a list, and produces as output the number of non-false values in the list.

(list-trues null) → **0**
(list-trues (list 1 2 3)) → **3**
(list-trues (list false (list 2 3 4))) → **1**

list-trues

- **Be *very* optimistic!** Since lists themselves are recursive data structures, most problems involving lists can be solved with recursive procedures.
- **Think of the simplest version of the problem**, something you can already solve. This is the base case. For lists, this is usually when the list is **null**.
- Consider how you would solve the problem using the result for a slightly smaller version of the problem. This is the recursive case. For lists, the smaller version of the problem is usually the **cdr** of the list.

list-trues

- **Be *very* optimistic!** Since lists themselves are recursive data structures, most problems involving lists can be solved with recursive procedures.
- **Think of the simplest version of the problem**, something you can already solve. This is the base case. For lists, this is usually when the list is **null**.
- Consider how you w… result for a slightly sm… is the recursive case. … the problem is usual…

```
(define (list-trues p)
  (if (null? p)
      0

  ))
```


list-trues

- **Be *very* optimis…** data structures, … solved with recu…
- **Think of the sim…** something you … For lists, this is u…
- **Consider how you would solve the problem using the result for a slightly smaller version of the problem. This is the recursive case. For lists, the smaller version of the problem is usually the cdr of the list.**

```
(define (list-trues p)
  (if (null? p)
      0
      (+ (if (car p) 1 0)
         (list-trues (cdr p)))))
```


Quiz

list-sum

Define a procedure, **list-sum**, that takes a list of numbers as input and outputs the sum of the numbers in the input list.

(list-sum (list 1 2 3)) → **6**
(list-sum null) → **0**

list-length

Define a procedure, **list-length**, that takes a list as input and outputs the number of elements in the input list.

(list-length (list 1 2 3)) → **3**
(list-length (list 1 (list 2 3))) → **2**
(list-length null) → **0**

Charge

- We'll repeat the Quiz on Friday if it seems too few people have done the readings well
- Problem Set 2: Due Monday
  - It is much longer than PS1, don't wait to get started
  - Help hours tonight in Olsson 001
- Friday: Recursive Procedures Practice