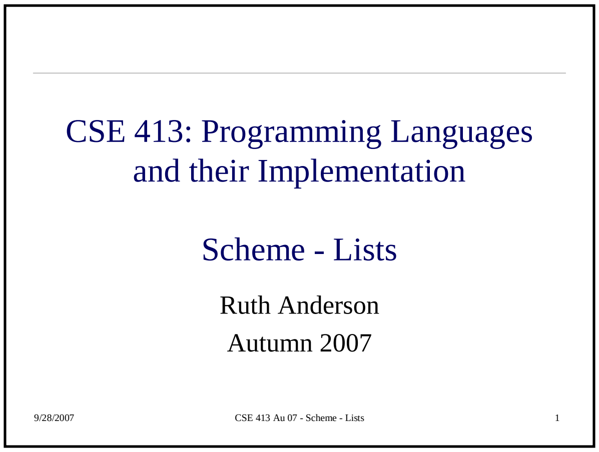
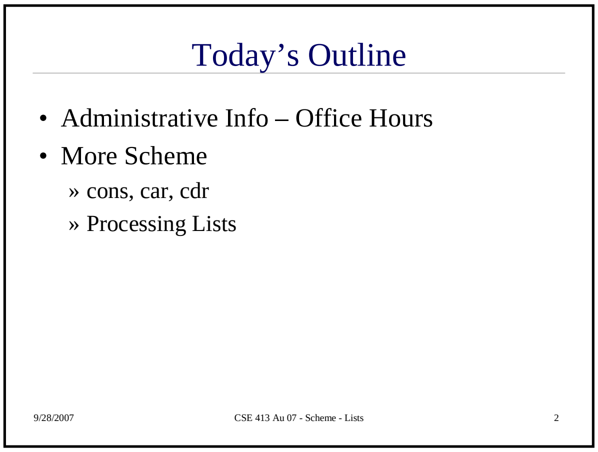


CSE 413: Programming Languages and their Implementation

Scheme - Lists

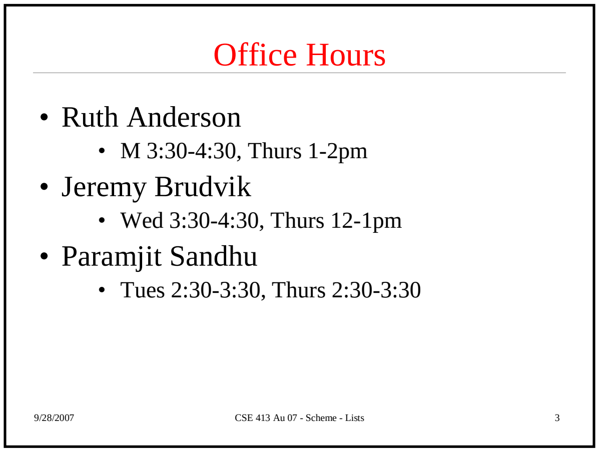
Ruth Anderson

Autumn 2007



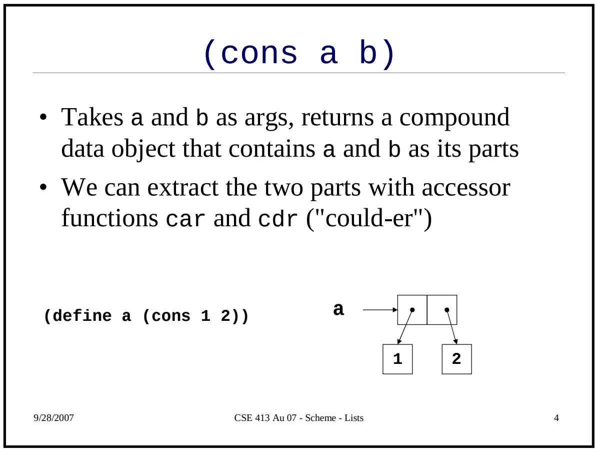
Today’s Outline

- Administrative Info – Office Hours
- More Scheme
  - » cons, car, cdr
  - » Processing Lists



Office Hours

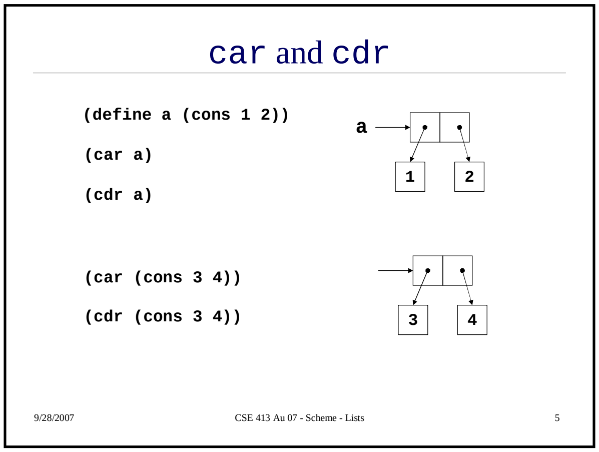
- Ruth Anderson
  - M 3:30-4:30, Thurs 1-2pm
- Jeremy Brudvik
  - Wed 3:30-4:30, Thurs 12-1pm
- Paramjit Sandhu
  - Tues 2:30-3:30, Thurs 2:30-3:30



(cons a b)

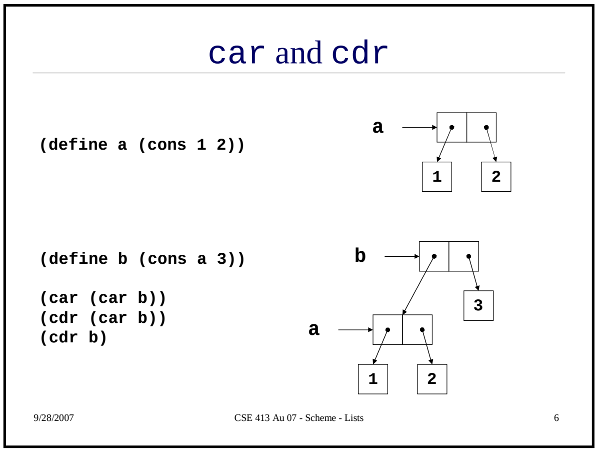
- Takes a and b as args, returns a compound data object that contains a and b as its parts
- We can extract the two parts with accessor functions car and cdr ("could-er")

```
(define a (cons 1 2))
```



car and cdr

```
(define a (cons 1 2))

(car a)

(cdr a)


(car (cons 3 4))

(cdr (cons 3 4))
```



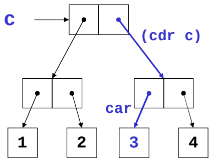
car and cdr

```
(define a (cons 1 2))


(define b (cons a 3))

(car (car b))
(cdr (car b))
(cdr b)
```

(car (cdr c))

```
(define c (cons (cons 1 2) (cons 3 4)))
```

```
(car (car c))
(cdr (car c))
(car (cdr c))
(cdr (cdr c))
```

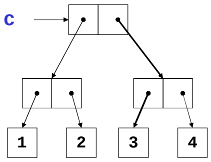


(cadr c)

- We can abbreviate the repeated use of car and cdr

```
(define c (cons (cons 1 2) (cons 3 4)))
```

```
(caar c)
(cdar c)
(cadr c)
(cddr c)
```

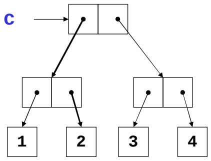


pair? predicate

- (pair? z) is true if z is a pair

```
(define c (cons (cons 1 2) (cons 3 4)))
```

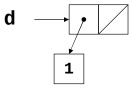
```
(pair? c)
(pair? (car c))
(pair? (cdr c))
(pair? (caar c))
(pair? (cdar c))
```



nil

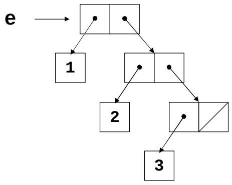
- if there is no element present for the car or cdr branch of a pair, we indicate that with the value nil
  - » nil (or null) represents the empty list '()
- (null? z) is true if z is nil

```
(define d (cons 1 '()))
(car d)
(cdr d)
(null? (car d))
(null? (cdr d))
```



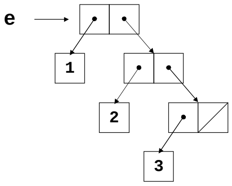
Lists

- By convention, a list is a sequence of linked pairs
  - » `car` of each pair is the data element
  - » `cdr` of each pair points to list tail or the empty list



List construction

```
(define e (cons 1 (cons 2 (cons 3 '()))))
```



```
(define e (list 1 2 3))
```

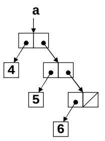
procedure `list`

(list *a b c* ...)

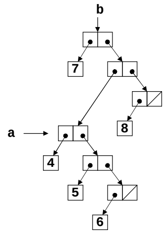
- `list` returns a newly allocated list of its arguments
  - » the arguments can be atomic items like numbers or quoted symbols
  - » the arguments can be other lists
- The backbone structure of a list is always the same
  - » a sequence of linked pairs, ending with a pointer to null (the empty list)
  - » the `car` element of each pair is the list item
  - » the list items can be other lists

List structure

```
(define a (list 4 5 6))
```

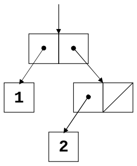


```
(define b (list 7 a 8))
```



Examples of list building

```
(cons 1 (cons 2 '()))

(cons 1 (list 2))

(list 1 2)
```



How to process lists?

- A list is zero or more connected pairs
- Each node is a pair
- Thus the parts of a list (this pair, following pairs) are lists
- A natural way to express list operations?

`cdr` down

```
(define (length m)
  (if (null? m)
      0
      (+ 1 (length (cdr m)))))
```

sum the items in a list

```
(add-items (list 2 5 4))
```