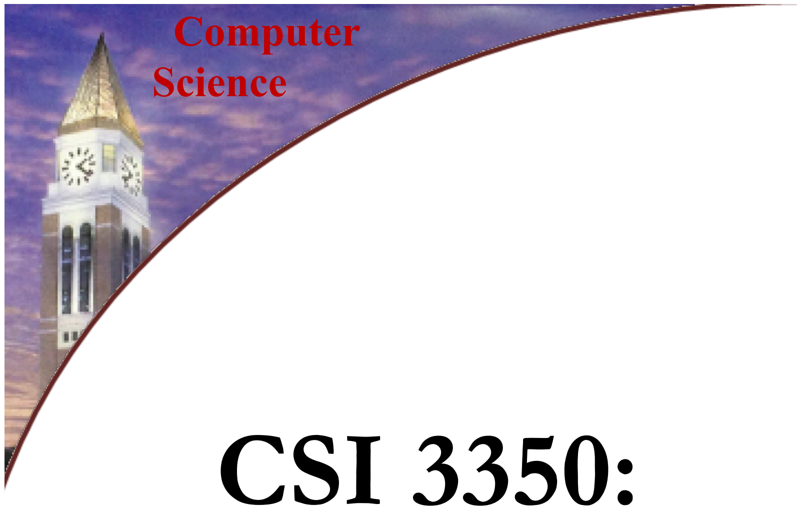
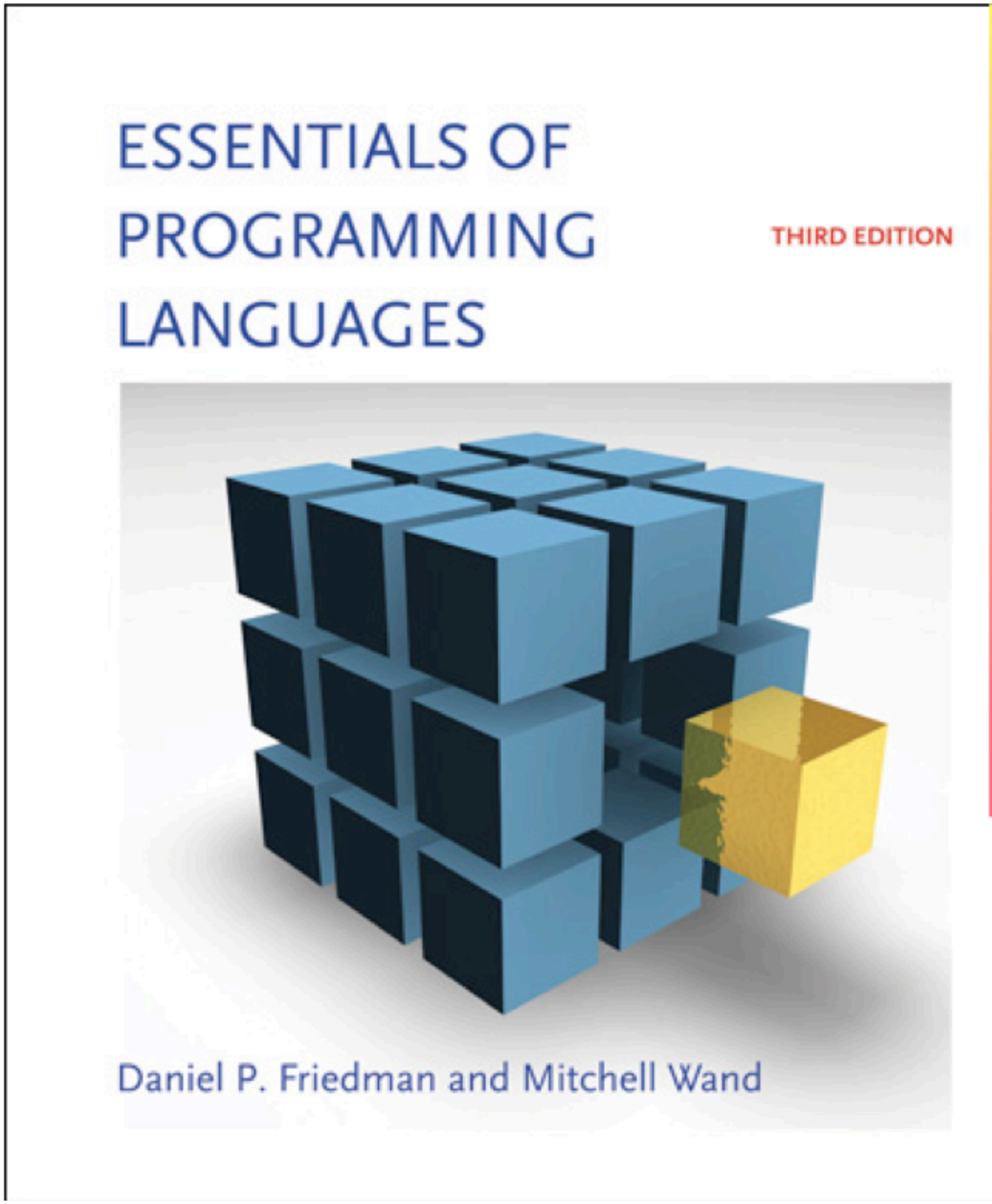


# CSI 3350: PROGRAMMING LANGUAGES

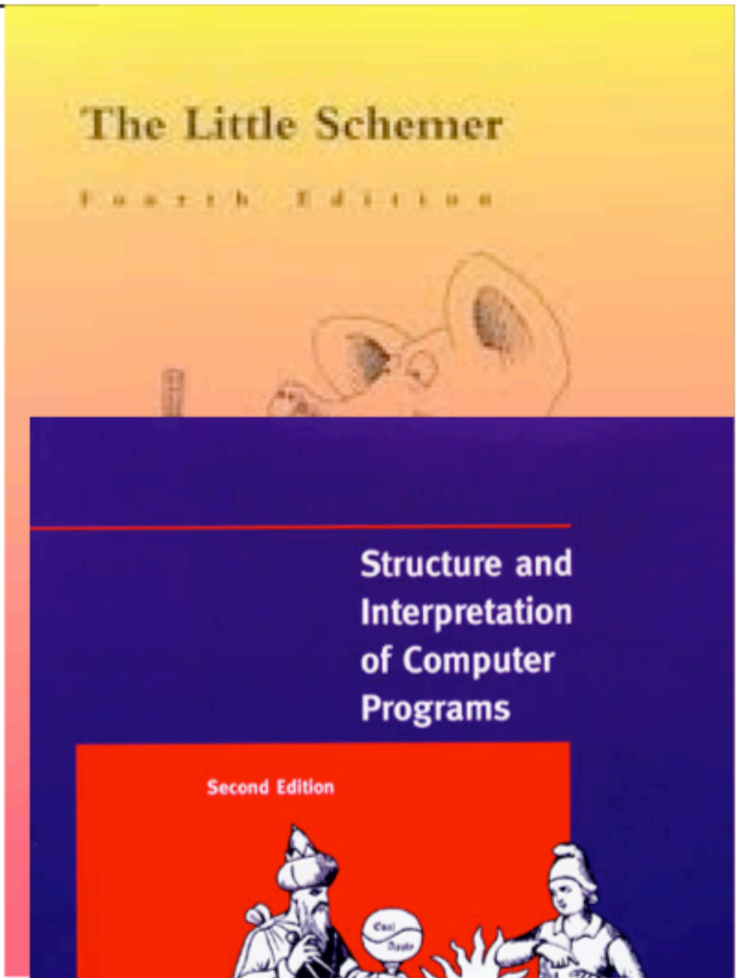
Department of Computer Science & Engineering

Oakland University

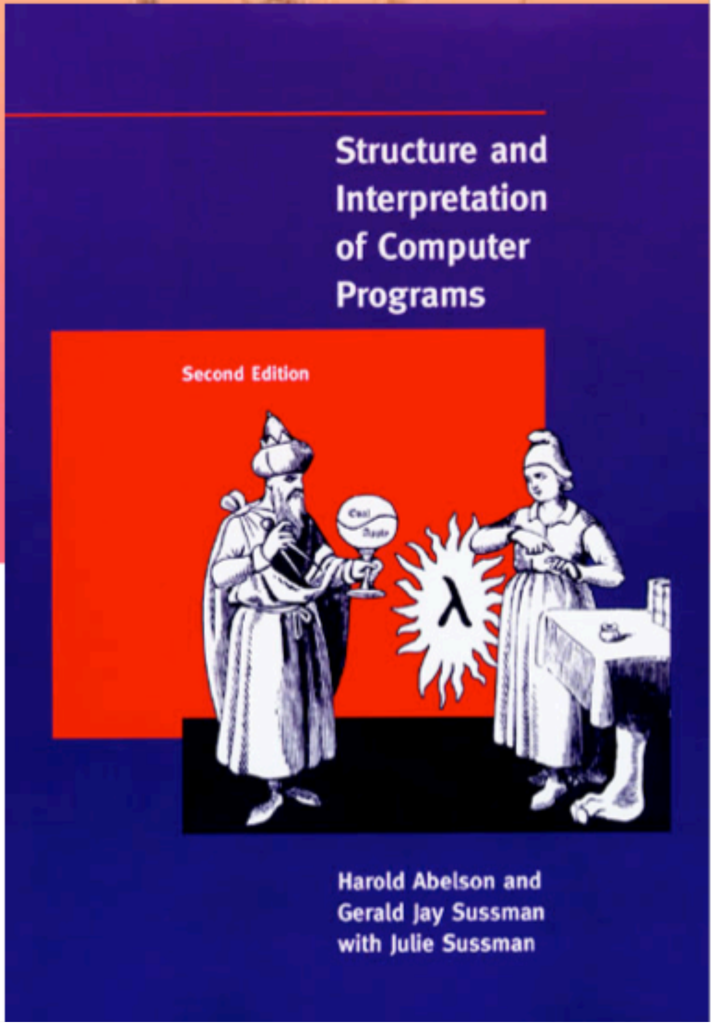

ESSENTIALS OF PROGRAMMING LANGUAGES

THIRD EDITION

Daniel P. Friedman and Mitchell Wand

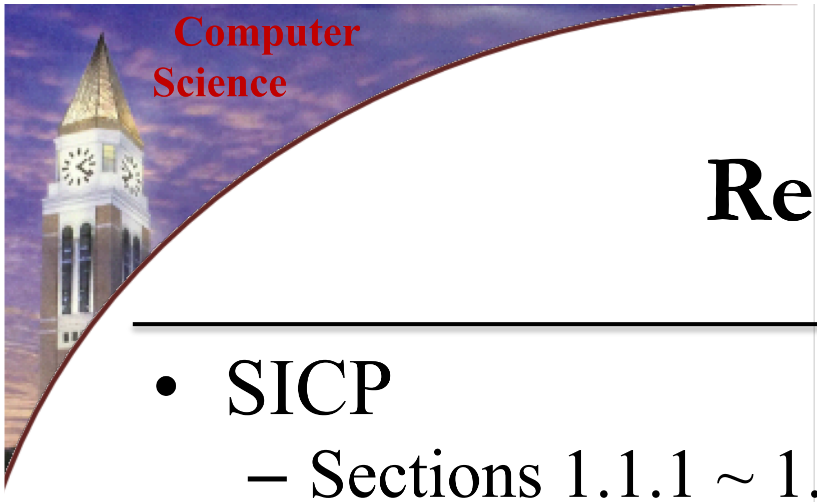

The Little Schemer

Fourth Edition


Structure and Interpretation of Computer Programs

Second Edition

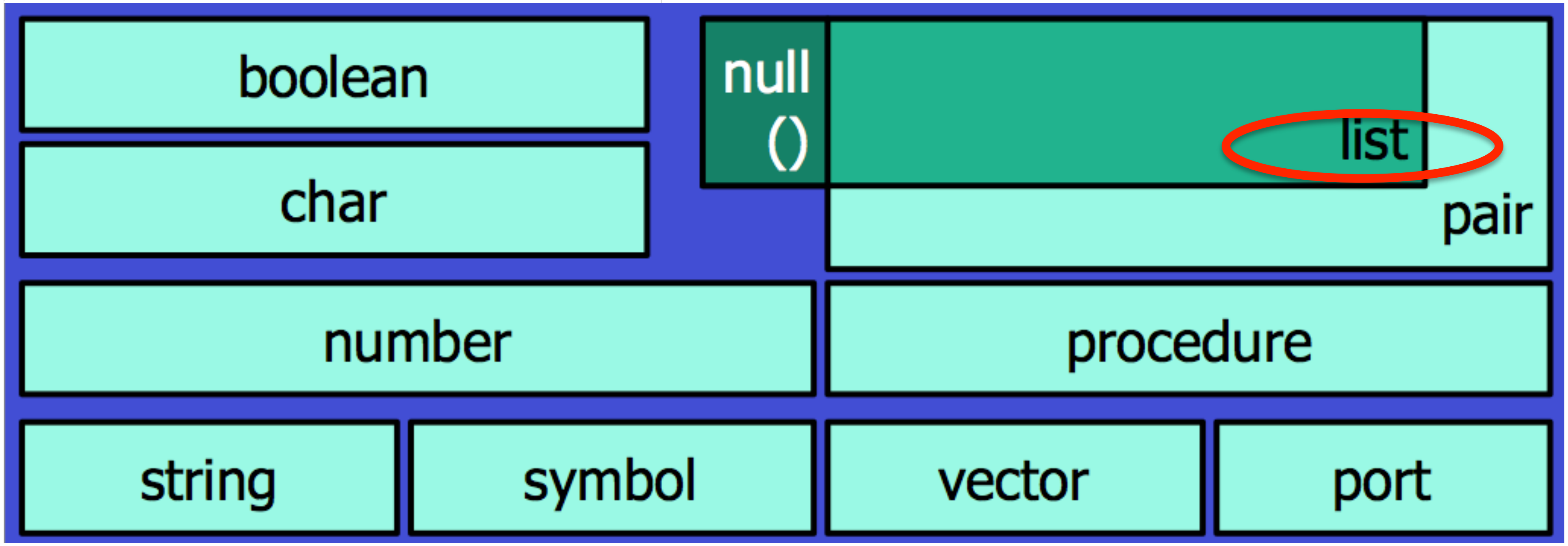
Harold Abelson and Gerald Jay Sussman with Julie Sussman

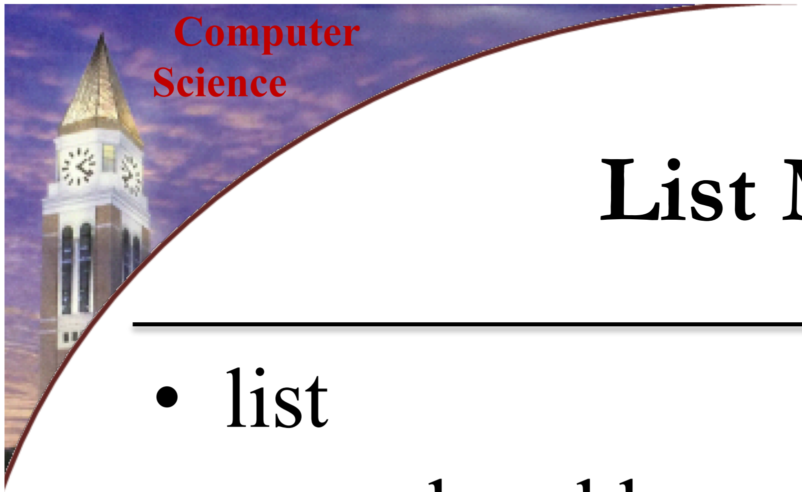


## Reading List

- SICP
  - Sections 1.1.1 ~ 1.1.6
  - Sections 2.2.1, 2.2.2 & 2.2.3
- The little Schemer
  - Preface p.xiii
  - Chap 1 ~ 3
- Revised Report on the Algorithmic Language Scheme
  - Section 1 [overview]
  - Section 6.1 – 6.3 [Standard Procedures]

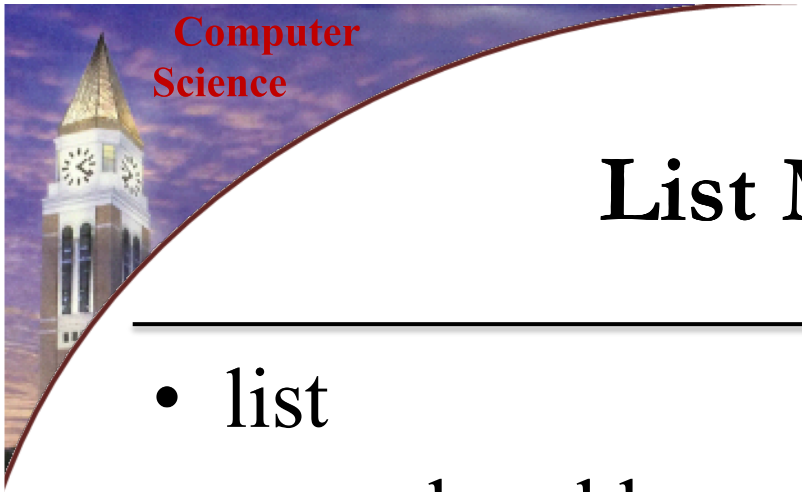
# Data Types in Scheme





## List Manipulation

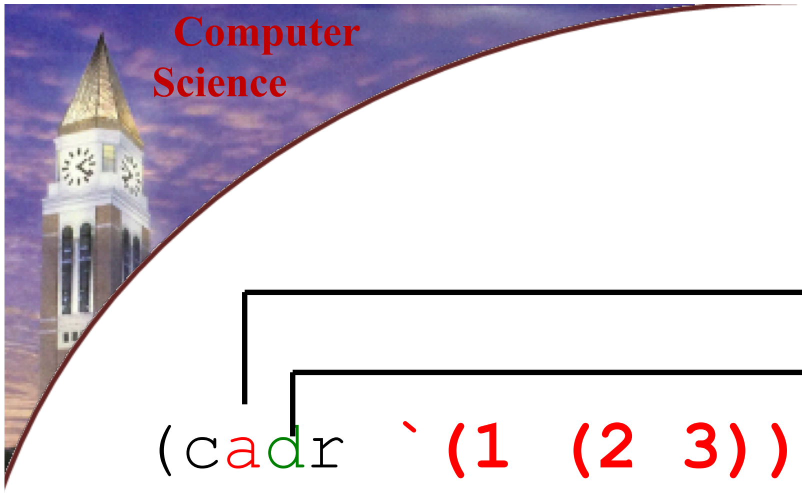
- list
- car, cdr, cddr, cadr etc
- first, second . . .
- length
- reverse
- append
- cons
- null?

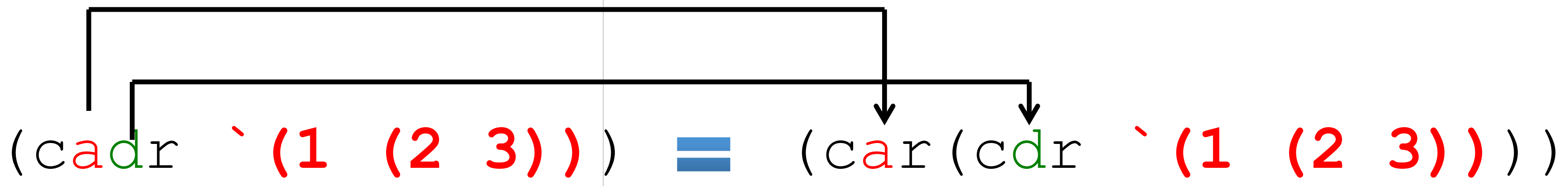


## List Manipulation

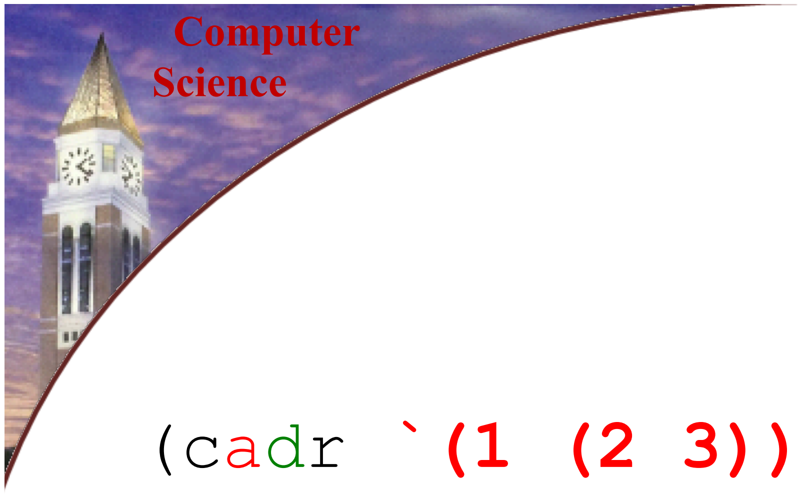
- list
- car, cdr, cddr, cadr etc
- first, second . . .
- length
- reverse
- append
- cons
- null?

**Google it -**
**search keyword: “DrRacket Scheme Library”**



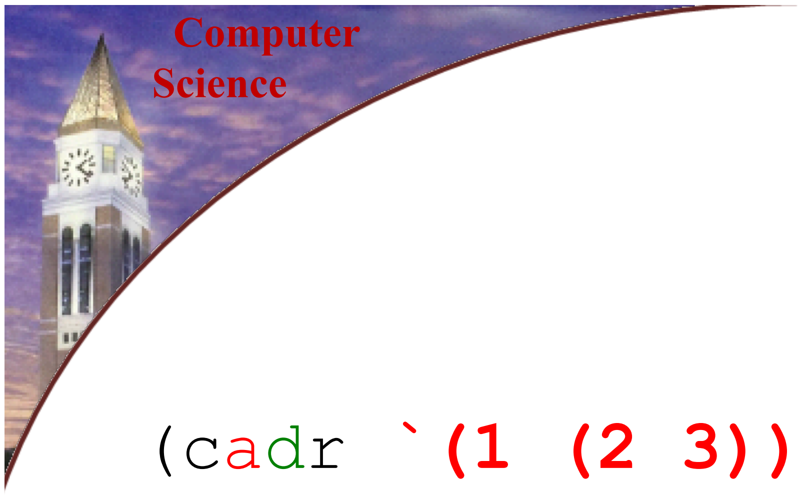


```
(cadr `(1 (2 3))) = (car(cdr `(1 (2 3))))
```



```
(cadr `(1 (2 3))) = (car(cdr `(1 (2 3))))
```

**?**



```
(cadr `(1 (2 3))) = (car(cdr `(1 (2 3))))
```

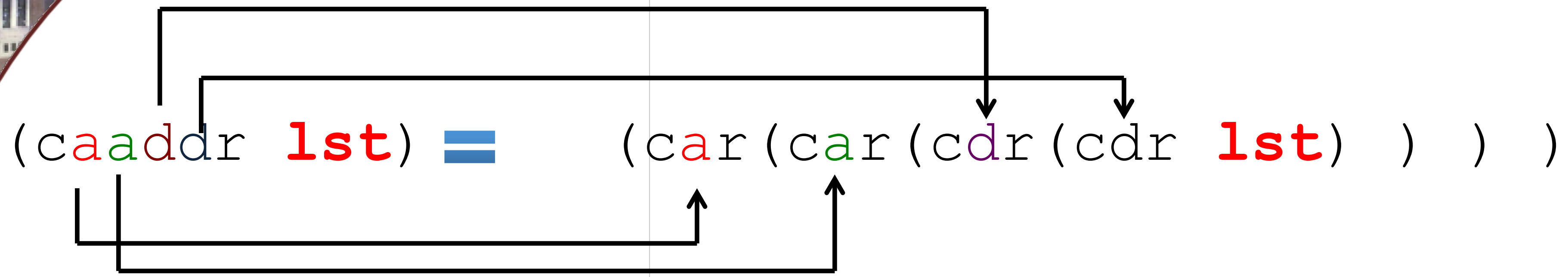
**`(2 3)**

`(caaddr lst)` =

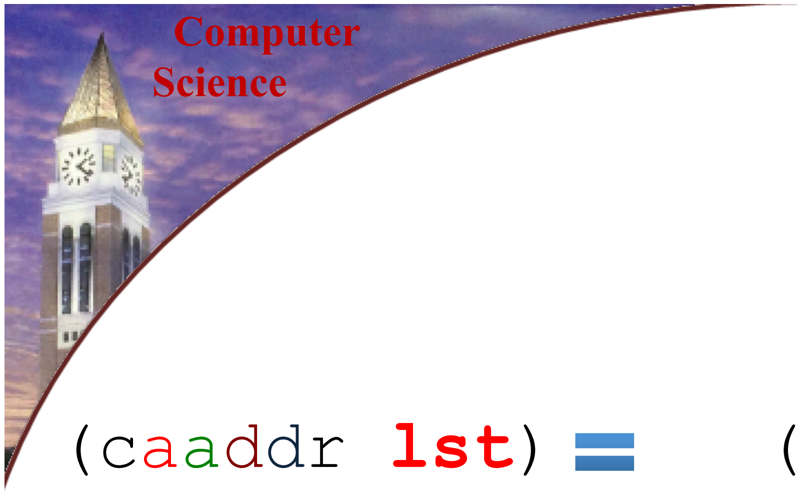
?

Computer Science

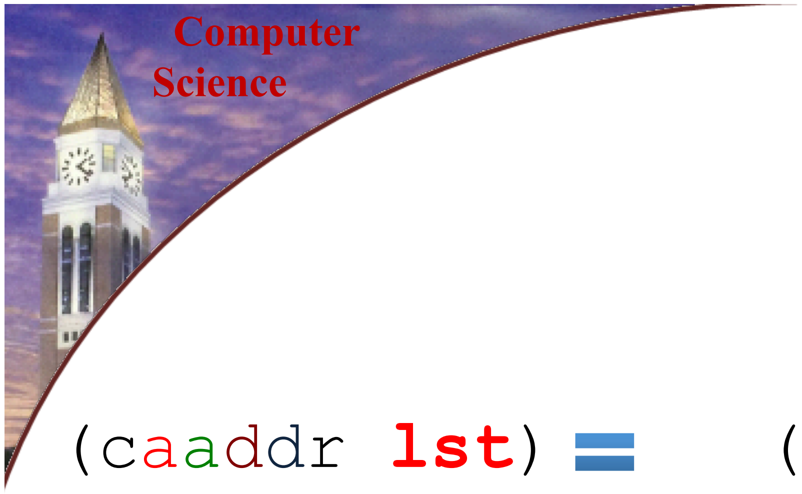
```
(caaddr lst) =    (car(car(cdr(cdr lst) ) ) )
```



```
(caaddr lst) = (car(car(cdr(cdr lst) ) ) )
```

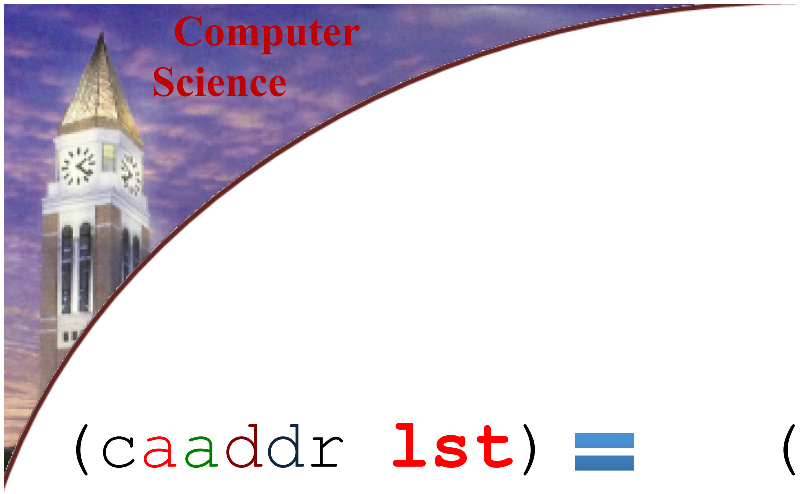


```
(caaddr lst) =    (car(car(cdr(cdr lst) ) ) )
```



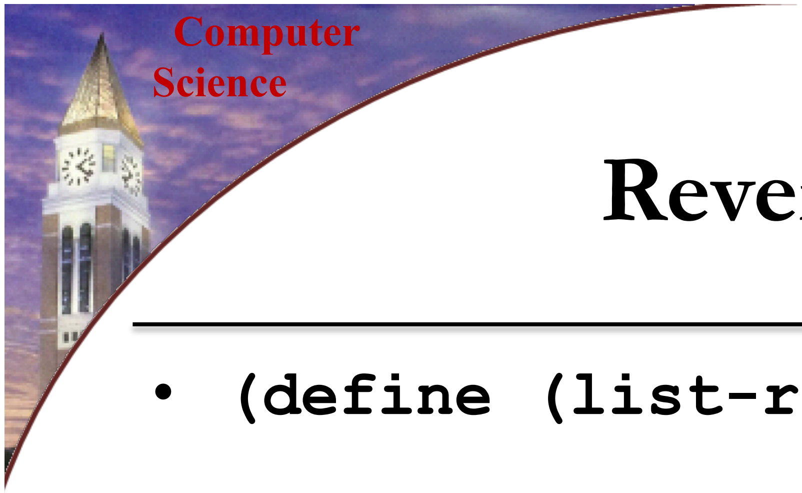
```
(caaddr lst) = (car(car(cdr(cdr lst) ) ) )
```

```
(caaddr '(1 2 3 4)) = ?
```



```
(caaddr lst) =     (car(car(cdr(cdr lst) ) ) )
```

```
(caaddr '(1 2 (3) 4)) =  ?
```



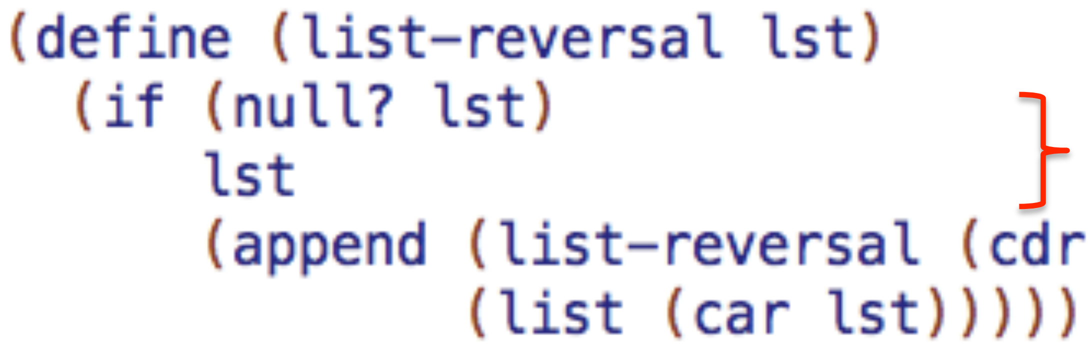
## Reversal of A List

- `(define (list-reversal lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(3 2 1)
'( 1 2 3 4)   => '(4 3 2 1)
```

## Reversal of A List

- `(define (list-reversal lst) ... )`



```scheme
(define (list-reversal lst)
  (if (null? lst)
      lst
      (append (list-reversal (cdr lst))
              (list (car lst)))))
```

Base case: empty `lst`

Recursive case

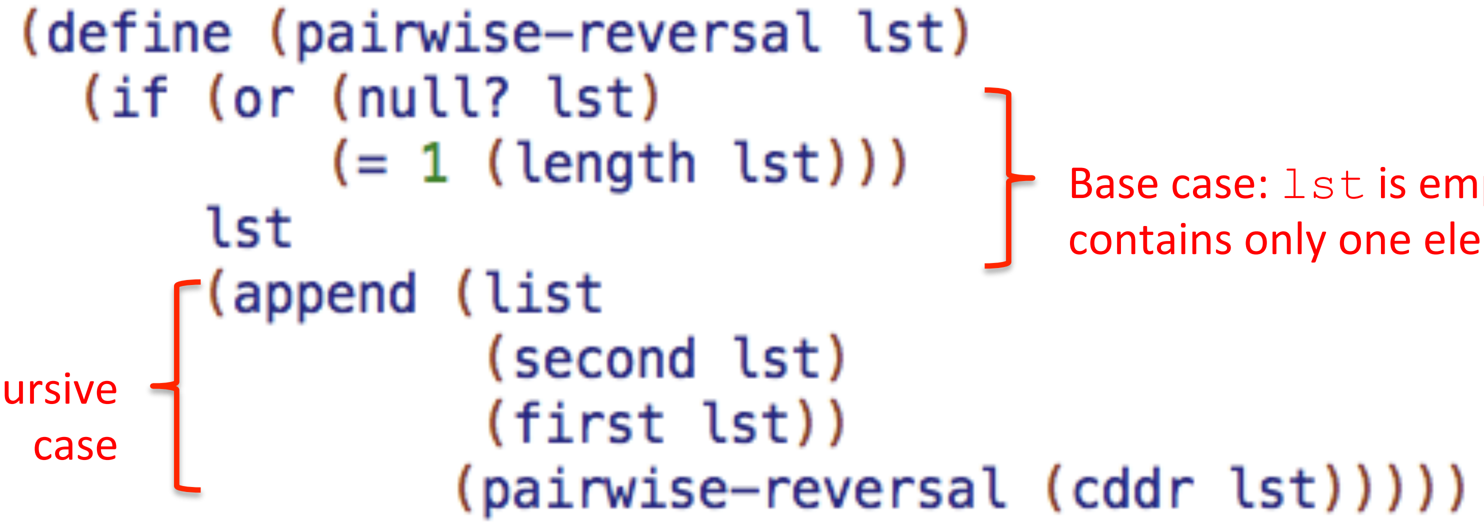
## Pairwise Reversal of A List

- `(define (pairwise-reversal lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(2 1 3)
'( 1 2 3 4)   => '(2 1 4 3)
```

## Pairwise Reversal of A List

- `(define (pairwise-reversal lst) ... )`



```
(define (pairwise-reversal lst)
  (if (or (null? lst)
          (= 1 (length lst)))
      lst
      (append (list
                (second lst)
                (first lst))
              (pairwise-reversal (cddr lst)))))
```

Base case: `lst` is empty or it contains only one element

Recursive case