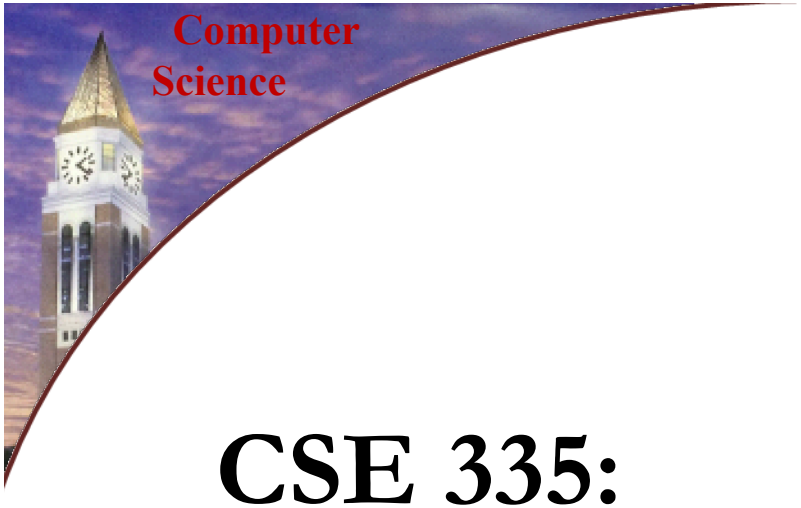


CSE 335: PROGRAMMING LANGUAGES

Department of Computer Science & Engineering

Oakland University

`if` and `cond`

(**if** condition
  consequent$_1$
  alternative
)

(**cond**
  (condition$_1$ consequent$_1$)
  (condition$_2$ consequent$_2$)
  . . .
  (condition$_n$ consequent$_n$)
  (**else** alternative)
)

Making Use of Number Types

Factorial

```
(define
    (fact n )
     . . .
)
```

Making Use of Number Types

Factorial

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

Assume: a is not greater than b

```
(define (sum-integers-between a b) ...)

> (sum-integers-between 2 5)
14
```

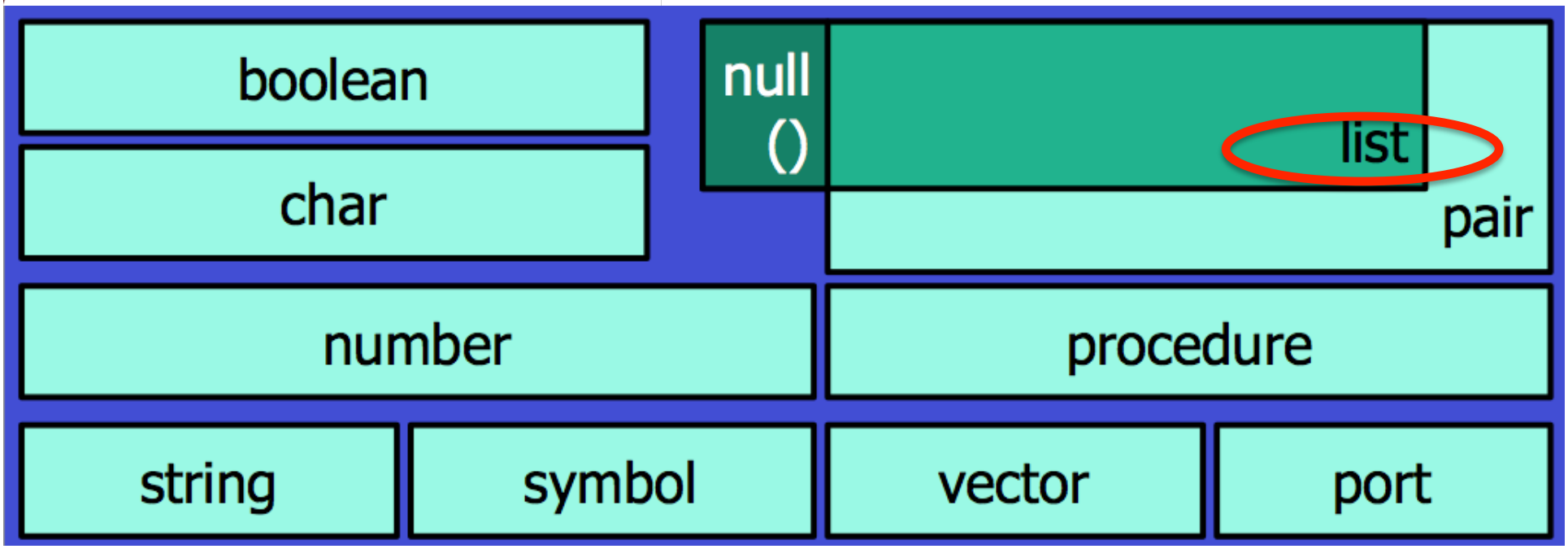
```
(define (sum-integers-between a b)
  (if (= b a)
      a
      (+ b (sum-integers-between a (- b 1)))))
```

```
(define (sum-integers-between a b)
  (if (= b a)
      a
      (+ b (sum-integers-between a (- b 1)))))
```

Base case

Recursive case

Data Types in Scheme



List Manipulation

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

List Manipulation

```
'( 1 2 3)

(car '( 1 2 3)) ➡ 1
(cdr '( 1 2 3)) ➡ '(2 3)
```

List Manipulation

```
'( 1 2 3)

(car '( 1 2 3)) ➡ 1
(cdr '( 1 2 3)) ➡ '(2 3)
(cadr '( 1 2 3)) ➡ 2
```

List Manipulation

```
'( 1 2 3)
(car '( 1 2 3)) ➡ 1
(cdr '( 1 2 3)) ➡ '(2 3)
(cadr '( 1 2 3)) ➡ 2
(cddr '( 1 2 3)) ➡ '(3)
```

List Manipulation

```
'( 1 2 3)
(car '( 1 2 3)) ➡ 1
(cdr '( 1 2 3)) ➡ '(2 3)
(cadr '( 1 2 3)) ➡ 2
(cddr '( 1 2 3)) ➡ '(3)
```

```
(cadr '( 1 (2 3)) ) ➡ ?
```

List Manipulation

```
(cadr `(1 (2 3)) )
```

List Manipulation

```
(cadr `(1 (2 3)) )
```

a compound function!

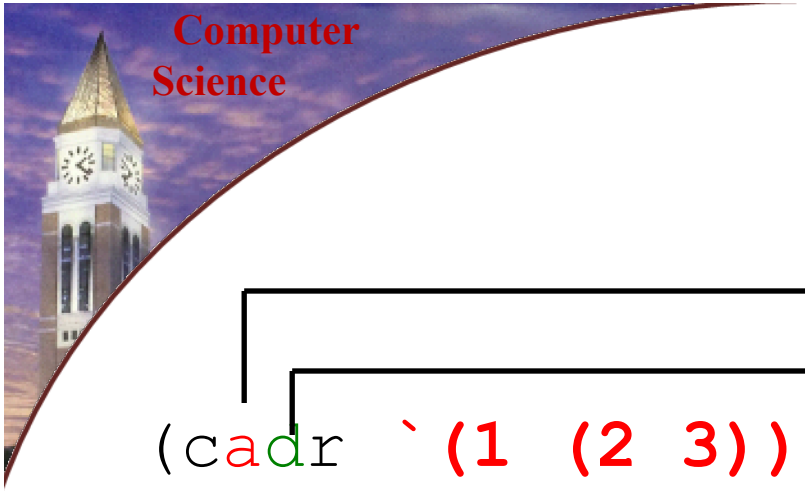
List Manipulation

```
(cadr `(1 (2 3)) )
```

List Manipulation

```
(cadr `(1 (2 3)) )
```

order of execution




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

```
(cadr lst)  ➡  (car(cdr)lst)
```

`lst` above refers any list, like `(1 (2 3))

```
(caaddr lst)
```

?


```
(caaddr lst)  ➡  (car(car(cdr(cdr lst) ) ) )
```

```
(caaddr lst)  ➡  (car(car(cdr(cdr lst) ) ) )
```

range function

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

zip function

```
(zip '(3 4 2)   '(5 9 7) )   ==> '((3 5) (4 9) (2 7))
(zip '(4 2)     '(9 7)    )   ==>  '((4 9) (2 7))

(zip '(2 3 1)   '(9 2)    )  ==>  '( (2 9) (3 2) )
```

zip function

(zip '() '(3 1 4 1 5 9)) ==> '()
(zip '(2 7 3 4) '()) ==> '()

} Base case

(zip '(3 4 2) '(5 9 7)) ==> '((3 5) (4 9) (2 7))
(zip '(4 2) '(9 7)) ==> '((4 9) (2 7))

zip function

```
(define (zip lst1 lst2)
  (if (or (null? lst1)
          (null? lst2))
      '()
      (cons (list (car lst1) (car lst2))
            (zip (cdr lst1) (cdr lst2)))
      )
  )
```

zip function

```
(define (zip lst1 lst2)
  (if (or (null? lst1)
          (null? lst2))
      '()
      (cons (list (car lst1) (car lst2))
            (zip (cdr lst1) (cdr lst2)))
      )
  )
```

Base case

Recursive case