

# Symbols and Quote

- So far, only atomic data we have used are numbers
- Now, we want to work with symbols as well
- We can have lists like

( 1 2 3 )

( cse232 cse233 cse411 )

# Symbols and Quote

- Now, we want to work with symbols as well
- We can have lists like

```
(( (c c) (3 3) )  
 ( (c d) (0 5) )  
 ( (d c) (5 0) )  
 ( (d d) (1 1) ) )
```

# Symbols

- Lists of symbols can look just like Scheme expressions

```
( * 4 ( + 9 8 ) ( - 4 2 ) )
```

```
(define (factorial n)
```

```
  (if (= n 1)
```

```
      1
```

```
      (* n (factorial (- n 1))))))
```

- (In fact, Scheme expressions *are* lists of symbols, that are read, evaluated, and whose values are printed by the interpreter)

# Symbols

- In our model of evaluation, however, all symbols are evaluated
- Scheme provides a special form to prevent evaluation
- `quote`
- The value of a `quote` special form *is* the thing quoted (Mantra #4)

# Symbols

- Example

```
1 ]=> (* 4 (+ 9 8) (- 4 2))
```

```
;Value: 136
```

```
1 ]=> (quote (* 4 (+ 9 8) (- 4 2)))
```

```
;Value: (* 4 (+ 9 8) (- 4 2))
```

# Symbols

- Example

```
1 ]=> (define (factorial n)
      (if (= n 1)
          1
          (* n (factorial (- n 1)))))
;Value: factorial
```

```
1 ]=> (quote
      (define (factorial n)
        (if (= n 1)
            1
            (* n (factorial (- n 1)))))
;Value: (define (factorial n) (if (= n 1) 1 (* n (factorial
```

# Symbols

- Scheme provides special shorthand for `quote`
- Use single quote `'` in front of what you want to quote

```
1 ]=> (quote (* 4 (+ 9 8) (- 4 2)))
```

```
;Value: (* 4 (+ 9 8) (- 4 2))
```

```
1 ]=> '(* 4 (+ 9 8) (- 4 2))
```

```
;Value: (* 4 (+ 9 8) (- 4 2))
```

# Symbols

- Note the syntax of `'`

```
1 ]=> (quote (* 4 (+ 9 8) (- 4 2)))
```

```
;Value: (* 4 (+ 9 8) (- 4 2))
```

```
1 ]=> ('(* 4 (+ 9 8) (- 4 2)))
```

Application of inapplicable object

```
(* 4 (+ 9 8) (- 4 2))
```

# Symbols

- **Important:** A symbol and its value are not the same!

```
1 ]=> (define a 1)
```

```
1 ]=> (define b 2)
```

```
1 ]=> (list a b)
```

```
;Value: (1 2)
```

```
1 ]=> (list 'a 'b)
```

```
;Value: (a b)
```

# Symbols

- A list of symbols can be created by quoting the conventional printed representation of lists

```
1 ]=> (define x '(a b c))
```

```
1 ]=> (car x)
```

```
;Value a
```

```
1 ]=> (cdr x)
```

```
;Value (b c)
```

# Symbols

- Scheme provides one other useful predicate just for symbols: `eq?`
- `eq?` takes two symbols as its arguments and tests whether they are the same

## Cooperative Exercise

- What would the interpreter print in response to

```
1 ]=> (list 1 2 3)
```

```
1 ]=> (list '1 '2 '3)
```

```
1 ]=> (car '((a1 b1) (a2 b2)))
```

# Symbols

- What would the interpreter print in response to

```
1 ]=> (list 1 2 3)
```

```
;Value: (1 2 3)
```

```
1 ]=> (list '1 '2 '3)
```

```
;Value: (1 2 3)
```

```
1 ]=> (car '((a1 b1) (a2 b2)))
```

```
;Value: (a1 b1)
```