Mutation

- set! allows an existing name-value binding to be changed.
Structuring data

• So far we’ve seen two types of values that we can bind to names:
  – numbers
  – functions

• two others are:
  – symbols
  – pairs
Symbols

• A symbol is an unevaluated name.
• Consider the following interaction:
  > (define x 12)
  > x
  12
• The x in the define is *not* evaluated. It is a symbol.
• The x in the second line is evaluated by looking up the value it is bound to in the current environment.
Using symbols

• We can bind a name to a symbolic value by quoting the symbol – quoting prevents evaluation:

  > (define fifi 'poodle)
  > fifi
  poodle

• The quote mark (') is syntactic sugar for the form (quote ...); the above is equivalent to:

  > (define fifi (quote poodle))
Pairs

- cons is a primitive function which builds a pair.
- A pair has two members.
- (cons 'a 'b) builds a pair
  - whose first member is the symbol a
  - whose second member is the symbol b
- cons is therefore a pair constructor.

- Graphically: 
  ![Pair Graph](a b)
accessors

• car returns the first member of a pair
• cdr returns the second member of a pair

> (define myPair (cons 'a 'b))
> (car myPair)
a
> (cdr myPair)
b
How is a pair displayed?

> (define myPair (cons 'a 'b))
> myPair
(a . b)

- Pairs are printed with a dot (.) separating the first (left) and second (right) components.
The REPL

- When interacting with a Scheme system, a Read-Eval-Print Loop (REPL) reads what you type, evaluates it, prints the resulting value, and loops back to repeat.
- The reader and printer parts of the REPL handle syntactic sugar (like converting 'a into (quote a)) and also printing things like lists.
Lists

• What is a list?
• A recursive definition:
  – the empty list () is a list
  – a pair whose cdr is a list is a list
• We can build a list using cons and ():
  > (define myList (cons 'a '()))
  > myList
  (a)
Wait a minute!!

> (define myPair (cons 'a 'b))
> myPair
(a . b)

> (define myList (cons 'a '()))
> myList
(a)
What's going on?

- Why did myList print as (a) rather than as (a . ())?
- In fact, they are entirely equivalent: the REPL strikes again. The printer formats pairs which cdr is a pair in a special way to make lists easier to read.
Printing lists

• \((a \cdot (b \cdot (c \cdot (d \cdot ()))))\) is printed as \((a \ b \ c \ d)\)
• Graphically the structure is:

```
  a  b  c  d
```

• This structure can be built in a variety of ways, some of which are shown on the next slide.
Choices…choices

- \((\text{cons} 'a (\text{cons} 'b (\text{cons} 'c (\text{cons} 'd '())))))\)
- \((\text{list} 'a 'b 'c 'd)\)
- \('(a\ b\ c\ d)\)
- \'(a . (b\ c\ d))\)
- \((\text{cons} 'a (\text{list} 'b 'c 'd))\)

and so on!