Association lists

• An association list is a list of pairs.
• Used to implement a look-up table of key-value pairs.
• The primitive assoc is used to search an association list, given a key.
• Example on next slide.
Association list example

> (define aList '(((fifi . poodle)  
   (fido . bulldog) (clifford . bigRed)  
   (lassie . collie)))
> (assoc 'fido aList)
(fido . bulldog)
> (assoc 'rufus aList)
#f
Mutating pair components

- set-car! mutates the value of the car of a pair
- set-cdr! mutates the value of the cdr of a pair

```lisp
> (define myPair (cons 'a 'b))
> myPair
(a . b)
> (set-car! myPair 'fred)
> myPair
(fred . b)
> (set-cdr! myPair 'wilma)
> myPair
(fred . wilma)
```
Lists in other languages

• A list is a recursive data structure
  – base case: an empty list is a list (called null, or (), in Scheme)
  – recursive case: a pair whose tail is a list is also a list (cons is pair op)
• Lists are fundamental also in Lisp, ML, Prolog, Erlang, etc.
• Lists in ML
  – list type is defined by cases:
    – [] is empty list constructor (a literal of type ‘a list)
    – :: is non-empty list constructor (a function of type ‘a * ‘a list -> ‘a list)
    – Example: 1 :: 2 :: [] produces list printed as [1,2]
      • can write list directly as [1,2]
• Lists in Prolog (not entire truth, but close enough for now)
  – same basic structure as in Scheme
  – [] is empty list constructor
  – [__] is non-empty list constructor
  – Example: [ 1 | [ 2 | [] ] ] produces list printed as [1,2]
    • can write list directly as [1,2]