# Sets

We will look at several ways to implement sets. We generally want sets to have unique elements -- an object x either is or is not an element of set S; it can't be an element of S more than once.

We will give 4 methods for each implementation --

- (make-set lat) which coverts a simple list of elements into a set.
- (element? x s) that returns #t or #f according to whether x is an element of set s.
- (union set1 set2)
- (intersection set1 set2)

Version 1:  We represent the set as a list of unique items; for (make-set  lat) we only need to remove the duplicate entries of lat.  Union and Intersection are easy.

Version II: We represent a set by a function that says if a particular element is a member of that set.

Version III: We represent a set by a Binary Search Tree that contains its elements.  Note that here we need element values that can be compared; we will assume our elements are numbers.

Now, what can you say about the three implementations?

- Which is more efficient?
- Which is easier to implement?
- Are there things you would like to do with sets that some implementations don't support?