Local definitions

- Organizational tool – group together a number of function, variable and structure definitions.

  need to switch to "intermediate student" language!

```
(local ( (def1 ...)
         (def2 ...) ... )
       (expression...) )
```

Local definitions

- The function, variable and structure definitions are available to each other, and to the expression.
- The definitions are not available outside of the local statement!

Example

```
(local
  ( (define (f x) (+ x 5))
    (define (g alon)
       (cond
         [(empty? alon) empty]
         [else
           (cons (f (first alons))
                 (g (rest alons)))])))
     (g '(1 2 3)))
```

Another Example

```
(local
  ( (define PI 3.141593)
    (define (area x)
       (* PI (* x x))))
  (area 12))
```

Better Examples – helper function

- local can be used when writing a function that requires a *helper function*.
  - *encapsulates* all the code (we don't want lots of "top level" helper functions).
- Easier to manage and to understand
  - all the code is in one place.

Countup function

- Create a function that creates a list of natural numbers from 1 to some parameter n.

```
(countup 4) => '(1 2 3 4)
```

old `countup` function

```
(define (countup n)
  (cond
   [(= n 1) (cons 1 empty)]
   [else
        (countup-helper 1 n)]))
```

old helper function

```
(define (countup-helper start end)
 (cond
  [(= start end) (cons end empty)]
  [else
   (cons start
         (countup-helper (add1 start) end))]))
```

New countup

```
(define (countup n)
 (local
  ((define (countup-helper start end)
    (cond
      [(= start end) (cons end empty)]
      [else
        (cons start
          (countup-helper (add1 start) end))])))
  (countup-helper 1 n)))
```

Exercise: Maximum number in a list of numbers

- Write a function that determines the maximum number in a list of numbers.

- Once it's working, see if you can't find a use for local:
  - the issue in this case is efficiency.

Exercise: Sorting a list of numbers

- given a list of number, produce an ordered list of numbers (ascending order).
- Obviously we want to use local!
  - just one "top-level" function
- Use this Strategy:
  - find smallest element – this goes first.
  - now sort the rest of the list.

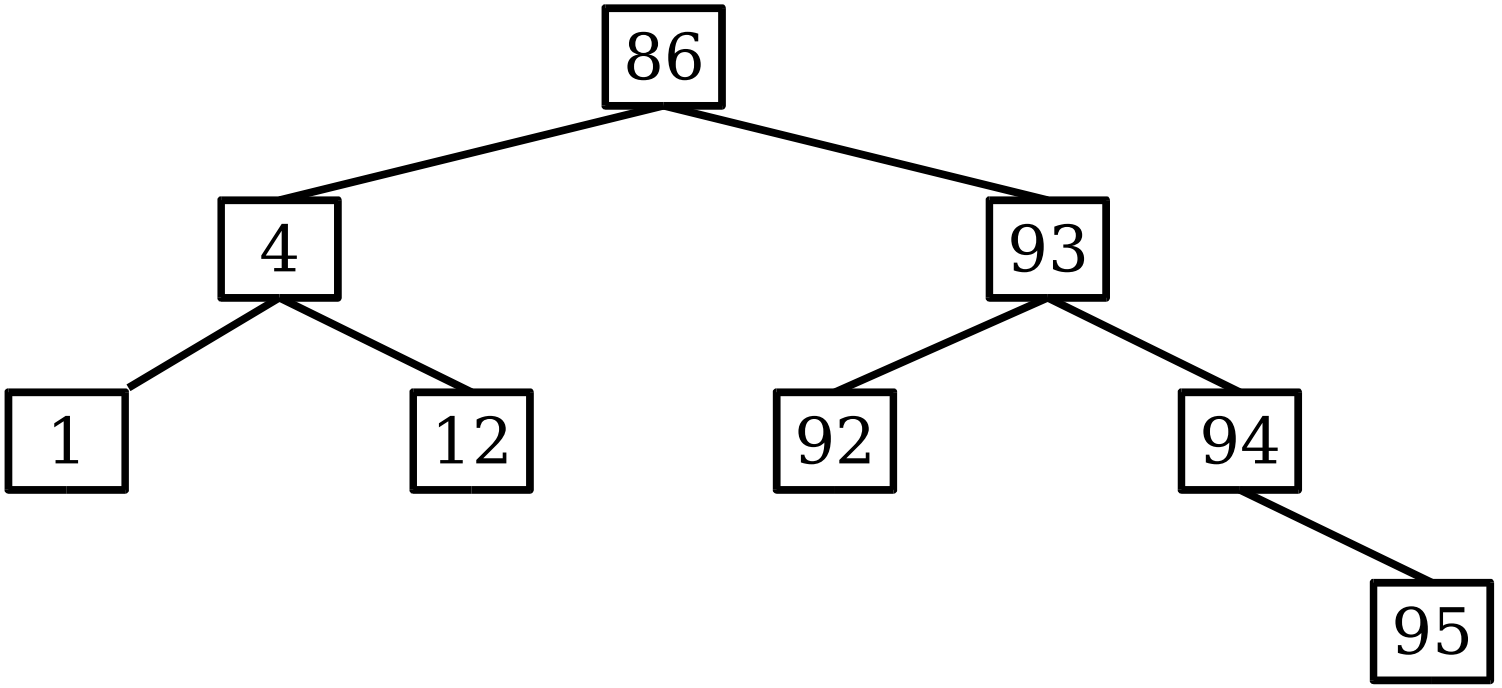
Tree Building

- Given a list of numbers (in any order), build a binary search tree.

```
(define-struct bsnode (num left right))
```

- Everything in left has num smaller, everything in right has num greater.

Binary Search Tree Example



Code we need to build BST

- order the elements?
- find one of the *middle* elements to use as the root if the tree?
  - consider what the tree would look like if we started with the smallest element at the root.
- find elements less than/greater than some number?
- build BST node given a number, and a list of smaller numbers, and a list of larger numbers?

When to use local (and when not)

- It is possible to do the whole thing in one function (using local).
  - the structure definition needs to be external (or it wouldn't make sense to create the tree – nothing else could use it!).
- If you write a function that might be useful in other situations, might want to consider makeing it a "top level" definition.
  - can't depend on the specifics of this problem.

Some test code:

```
(make-bst '(5 2 1 3 4)) =>
(make-bsnode 3
  (make-bsnode 1 empty
    (make-bsnode 2 empty empty))
  (make-bsnode 4 empty
    (make-bsnode 5 empty empty)))
```