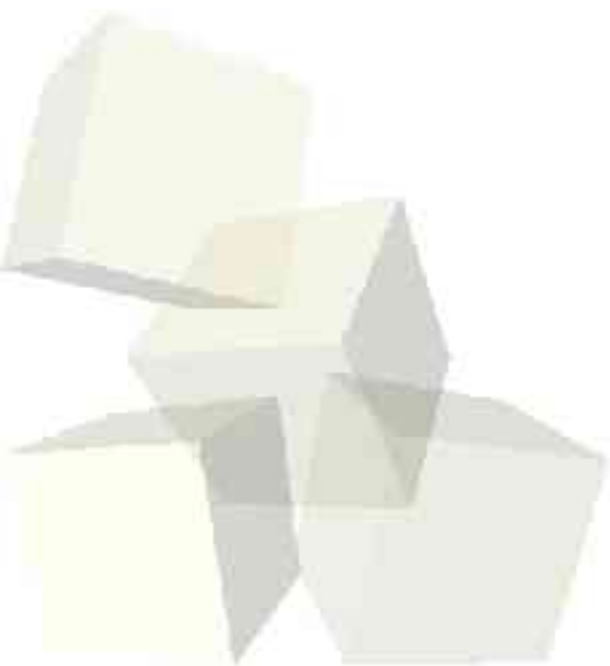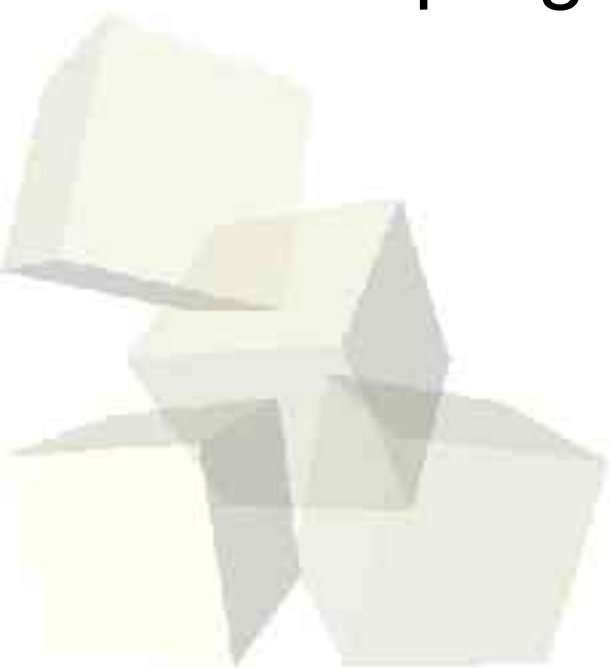# Interactive Programming

9-22-2004

Opening Discussion

- What did we talk about last class?
- Do you have any questions about assignment #3. Today's lecture will obviously help you a lot with what to do for that program.

Code for Fast Sorts

- I wrote code for the faster sorts, merge-sort and quicksort. Let's take at look at them.
- These sorts are naturally recursive. Let's see how that comes out in Scheme.
- What parts are somewhat unnatural about them?
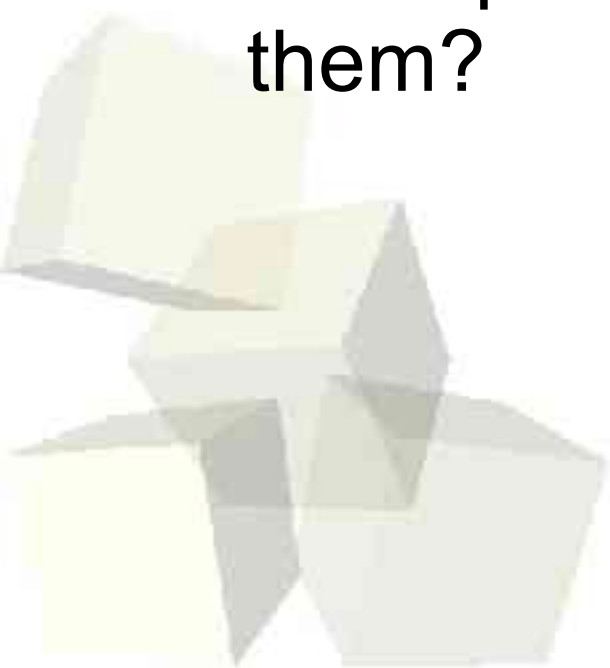
How do Sorts in Scheme Compare?

- I asked last time on the minute essay about the relative efficiency of these sorts compared to how you would do them in an imperative language.
- The number of comparisons is basically the same. However, the nature of the lists means that we use a lot more memory in cons cells. If we used vectors we would likely get code nearly identical to the imperative languages.
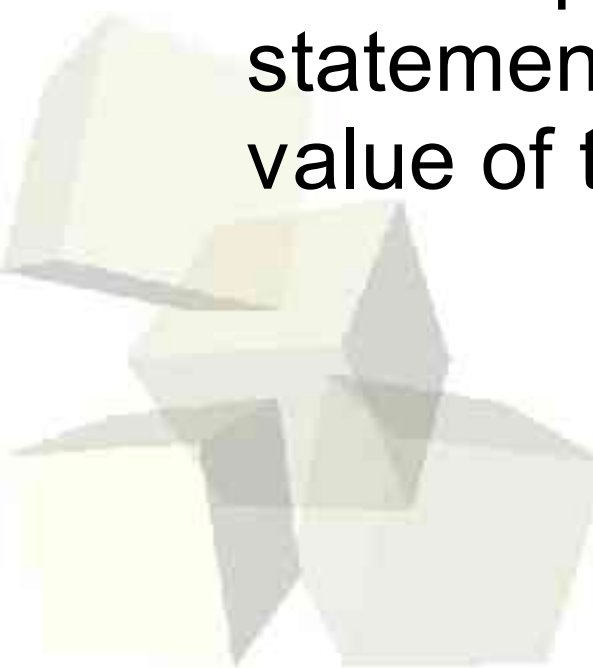
Strings

- We have seen that we can make strings in Scheme like in most other languages with double quotes. There are also functions to help us deal with strings.
  - String-length
  - String-append
  - Substring – give first and last index. Last is exclusive.
  - symbol->string – converts a symbol name to a string.
  - String=? – case sensitive string compare.
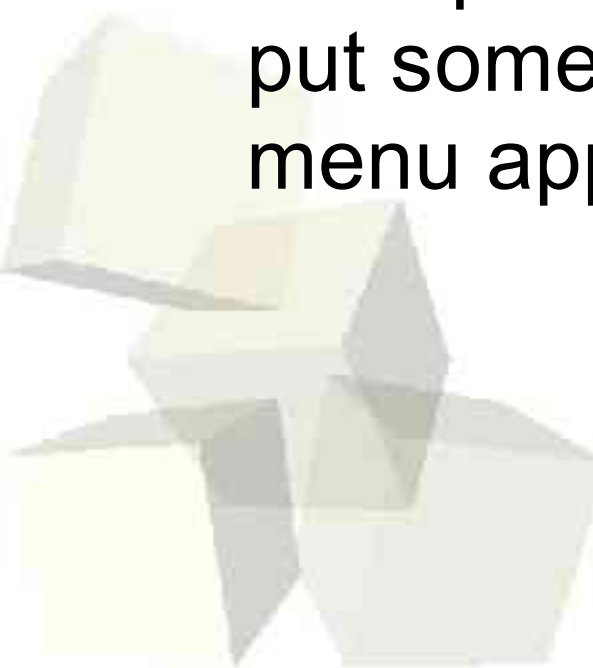 - String-ci=? – case insensitive compare.

Implicit begin

- We have seen the use of begin to do statements that have side effects and aren't returned.
- It happens that lambda, let, letrec, and cond have implicit begins. We can put multiple statements in their body and it returns the value of the last one.

Input in Scheme

- The read function in Scheme can be used to read user input. You can input valid Scheme expressions including strings in quotes and lists.
- Let's pull up the inventory program and try to put some logic in it so that we can have a menu application that uses user input.

Minute Essay

- Input and output are nice for making programs usable. However, they are fundamentally non-functional. Describe why this is.
- The ML text book is now in the book store so you can get it at your convenience.