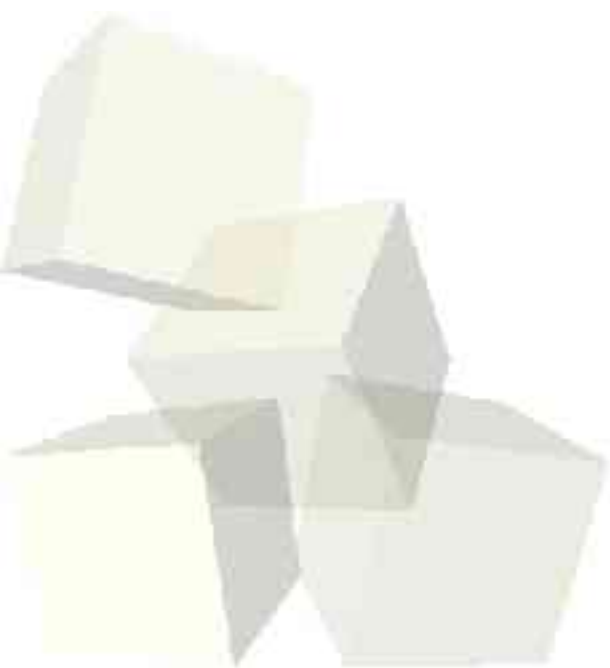




# Lists and Tuples

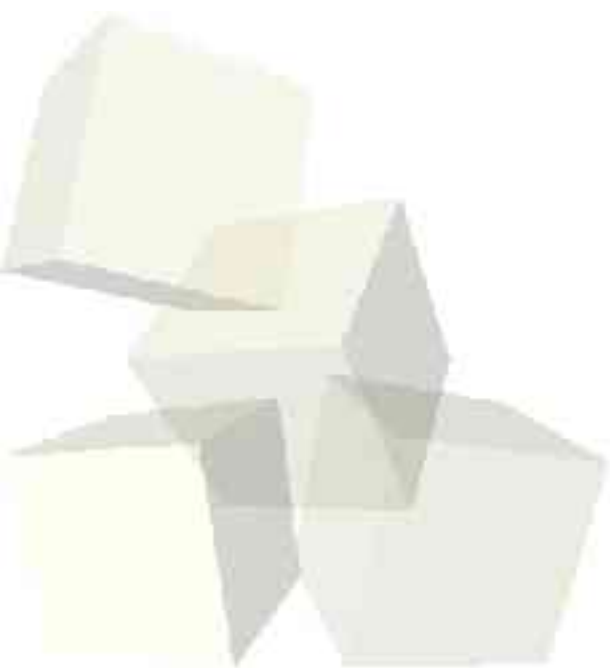
10-13-2004





# Opening Discussion

- What are some of the things we talked about last class? Yes, it was a very long time ago.
- ML does not have variables in any true sense. Let's look at a picture to see the difference.





# Tuples

- ML has a construct called a tuple unlike anything in Scheme. A tuple is a type that groups other types together. A tuple is written as a comma separated list in parentheses.
- So (1,"a",3.0) is a tuple of type `int * string * real`. Note the number of elements is fixed.
- We can access the elements of a tuple with a syntax like the following `#1name`, `#2name`, etc. Note it is 1 referenced.



# Lists

- Like Scheme, ML makes extensive use of lists, but they have some differences.
- An ML list is a comma separated set of elements in square brackets. All the elements must have the same type.
- So [1,2,3] is of type int list.
- If you try to make a list [1,2,3.0] it will be an error because the types are different.
- The number of elements in a list is not fixed, just the type.

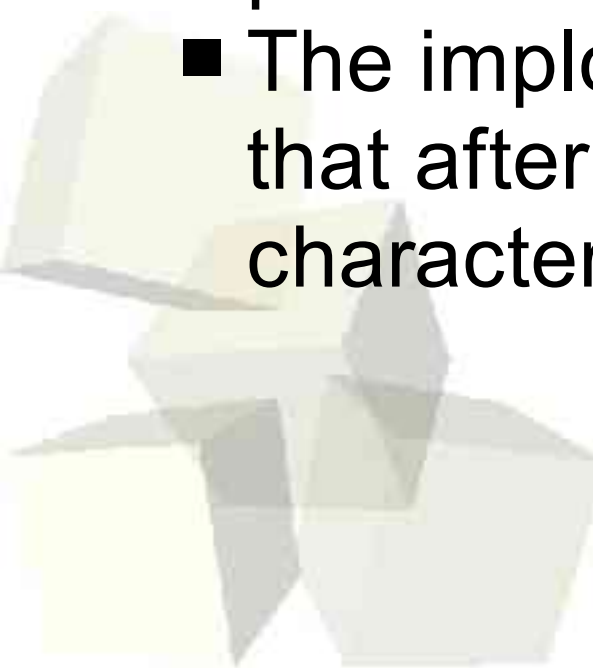


# Heads, Tails, Cons, and Concatenation

- Like in Scheme we have functions to pull lists apart and other functions to put them together.
- Instead of `car` and `cdr` ML uses the more intuitively named `hd` and `tl` for head and tail. Note the types. If we have an `int` list then `hd` is an `int` and `tail` is an `int` list.
- `nil` or `[]` represents the empty list.
- We cons a list with the infix operator `::`, so `1::2::nil = [1,2]`.
- The `@` operator appends lists.



# From Strings to Lists and Back

- ML provides two functions that can be used to help play with strings.
  - The `explode` function takes a string and returns a list of characters so that you can parse through the individual elements.
  - The `implode` function does the inverse so that after you have altered the list of characters you can get back to a string.
- 



# Basics of Functions

- The simple way to define a function in ML is with the keyword fun.
  - `fun <identifier> (<parameters>)=<expression>;`
- This creates a function that goes by the provided identifier.
- Note that ML infers the type of the function and shows us that type. It should be noted that `->` binds right to left. This will matter later on.
- We only declare types if we don't want the default. We do that with a colon and the type.



# Calling Functions

- We call a function much like we would in C though in many cases the parentheses aren't needed.
- When we write a function of multiple arguments it actually takes a single argument that is a tuple. In that case we need the parentheses to bind the tuple together.
- If a function is defined to use an outside variable, it gets the value of the variable at the time it is defined, not the calling time.





# Comments

- You will want to put at least some comments in your code. At the very least, your name is required.
- In ML we specify comments with (\* and \*).
- They can be nested the way that parentheses are nested. This is very helpful when you have to comment out chunks of code that include other comments.





## Minute Essay

- Write two functions. One to cube a real and one that takes a string and returns a string that contains the input twice.
- On the links page I added a link to a programming language shootout. They do some little speed tests on a number of languages. The tests aren't perfect, but you should note how well ocaml and mlton (an ML implementation) do on many of the tests.