Announcements:

- Everyone should now have a section
- OCaml demo sessions Friday, time and place TBA
- PS1 due Tuesday 11:59PM
- Quiz #1 on Thursday, first 10 minutes of class

- Square example and substitution
  - `let square = fun z -> z*z`
- Anonymous functions
  - Why should everything have a name?
    - Values don’t!
- In ML, functions are “first class objects”
  - Don’t discriminate against them!
Namespace management: scope, modules, etc.

Lexical scope
  - Very important to understand which variable an identifier refers to
  - Source of many subtle bugs
  - Common prelim question

Let binds variables to values with a scope
  - `let id = e1 in e2`
  - Evaluate e1. Replace id in e2 by this value. The result of evaluating the new e2 is the value of the let expression.
    - Almost no exceptions to the substitution (string example, e.g.)
  - Nested lets have a “block structure”

Example:
  - `(let x = 3 in x*2) + x`
  - Think of let as “make this substitution within this block”
  - EQUATIONAL REASONING
  - How to think about the top-level loop?

Parallel binding via and
  - `let x = 3 and y = 7 in x+y`
  - `let x = 3 and y = x+4 in x+y`
  - `let x = 3 in let y = x+4 in x+y`
  - Can be dangerous, but sometimes very useful
- **Defining functions**
  - Most important elements of the namespace
  - Lots of subtleties
  - Example: `let f x = e1 in e2`
    - Scope of x is e1
    - Scope of f is e2
    - Good quiz question...
  - **Syntactic sugar** for
    - `let f = fun x -> e1 in e2`
  - Useful to remember this equivalence
  - There is another equivalent form we will get to soon: currying!

- Side note: can also use modules for namespace management
  - `String.length` vs. `open String followed by length`
  - Some modules are open by default, such as `Pervasives`
    - Why not `String`??

- Also note: functions take exactly 1 argument
  - `let f(x,y) = x + y;`
  - `let z = (1,2)`
  - `f(z)`
Recursive function definitions

- Suppose we try to write factorial using let. [Try it]
- Doesn’t work. Why?
- We need instead to use let rec instead
- let rec fact z = if z = 0 then 1 else z*fact(z-1)
in fact 3
- Can be used for mutually recursive functions!
  let rec even x = x = 0 || odd (x-1)
  and odd x = not (x = 0 || not (even (x-1)))
in
  odd 3110
- This can be very powerful and easy to abuse