Fall 2012 Announcements:

- Everyone should now have a section
- No section Monday. Section rooms (but not time slots) are slightly in flux, we will split the 2:30 section but probably not the others
- PS1 due Thursday 9/6 at 11:59PM
- Quiz #1 probably next week, first 10 minutes of class

- Square example and substitution
    - Informal introduction about "how to think in OCaml"
    - `let square = fun z -> z*z in square 1+2`
    - Amusing fact: this actually returns 3 (put parens in to return 9)

Fun with lists

```
let rec inclist (lst: int list) =
  match lst
  with
   [] -> []
  | h::t -> h+1::inclist(t)


let square = fun z -> z*z



let rec sqlist (lst: int list) =
  match lst
  with
   [] -> []
  | h::t -> square(h)::sqlist(t)




let rec imap (f:int->int) (lst: int list) =
  match lst
  with
   [] -> []
  | h::t -> (f h)::(imap f t)

(* Exercise: write inclist and sqlist using imap, i.e. without recursion *)
```

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
- Variable declarations in OCaml bind variables within a **scope**, the part of the program where the variable stands for the value it is bound to
- Let (common case) binds variables to values within its body
  - `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? Everything you typed before is a giant let!

- Parallel binding via and
  - `let x = 3 and y = 7 in x+y (* 10 *)`
  - `let x = 3 and y = x+4 in x+y (* error *)`
  - `let x = 3 in let y = x+4 in x+y (* 10 *)`
- Some examples:

```
let x = 1 + 3 in (let x = 42 in x + 1) - x

let x = 1 + 3 in let y = x + 1 in let x = x + x in x*y

(* even more fun! *)
let x = 3 + 2 in
  let y = x + 1 in
    let x = fun x -> x + y in
     x(y)
```

- Can be dangerous, but sometimes very useful
  - The person who suffers from your clever code can be you!

- 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!