Today’s Agenda

• Mutual Recursion
• Module System Example
  • Namespace Organization
  • Preserving Invariants
• Practice with Currying and High Order Functions
Mutual Recursion

- What if we need function f to call g, and function g to call f?
- This is a common idiom

```plaintext
fun earlier x = 
  ... 
  later x 
  ... 

fun later x = 
  ... 
  earlier x 
  ...
```

Unfortunately this does not work 😞
Mutual Recursion Workaround

• We can use higher order functions to get this working
• It works, but there has got to be a better way!

```haskell
fun earlier f x =
  ...
  f x
  ...

fun later x =
  ...
  earlier later x
  ...
```
Mutual Recursion with **and**

- SML has a keyword for that
- Works with mutually recursive **datatype** bindings too

```sml
fun earlier x = 
  ...
  later x
  ...

and later x = 
  ...
  earlier x
  ...
```
Module System

• Good for organizing code, and managing namespaces (useful, relevant)
• Good for maintaining invariants (interesting)
Interesting Examples of Invariants

• Ordering of operations
  • e.g. insert, then query

• Data kept in good state
  • e.g. fractions in lowest terms

• Policies followed
  • e.g. don't allow shipping request without purchase order
Currying and High Order Functions

• Some examples:
  • List.map
  • List.filter
  • List.foldl