Hi! I’m Justin ^_^  

Senior in CS  
Love PL!!!  
Huge Haskell fan  
Third time TAing for 341  
Vim!

Today  

Motivation for this course  
SML workflow, errors, and booleans

What’s so exciting about this class and why should I care?  

Functional programming!  
Completely different style from what you’re probably used to  
No loops, only recursion, no mutation etc...  
Concise code!!!  
Lot of features present in other languages  
May never write a line of the langs we cover again  
But features from FP languages have seeped into “mainstream” languages  
Will highlight

What does this do?  

```haskell
let f = filterM $ const [True, False]
```

Java solution:

```java
public static <T> Set<Set<T>> powerSet(Set<T> originalSet) {
    Set<Set<T>> sets = new HashSet<Set<T>>();
    if (originalSet.isEmpty()) {
        sets.add(new HashSet<T>(null));
        return sets;
    }
    List<T> list = new ArrayList<T>(originalSet);
    T head = list.get(0);
    Set<T> rest = new HashSet<T>(list.subList(1, list.size()));
    for (Set<T> set : powerSet(rest)) {
        Set<T> newSet = new HashSet<T>(){
            set.add(head);
            set.addAll(set);
        }
        sets.add(newSet);
        sets.add(set);
    }
    return sets;
}
```
Using \textit{use} \texttt{use "foo.sml";}

\begin{itemize}
\item Enters bindings from the file \texttt{foo.sml}
  \begin{itemize}
  \item Like typing the variable bindings one at a time in
    sequential order into the REPL (more on this in a
    moment)
  \end{itemize}
\item Result is (\texttt{()}) bound to variable \texttt{it}
  \begin{itemize}
  \item Ignorable
  \end{itemize}
\end{itemize}

\begin{quote}
\textbf{Shadowing of Variable Bindings}
\end{quote}

\begin{verbatim}
val a = 1; (* a -> 1 *)
val b = a * 10; (* a -> 1, b -> 10 *)
val a = 2; (* a -> 2, b -> 10 *)
\end{verbatim}

\begin{itemize}
\item Expressions in variable bindings are evaluated “eagerly”
  \begin{itemize}
  \item Before the variable binding “finishes”
  \item Afterwards, the expression producing the value is irrelevant
  \end{itemize}
\item Multiple variable bindings to the same variable name, or “shadowing”, is allowed
  \begin{itemize}
  \item When looking up a variable, ML uses the most recent binding by that name in the current
    environment
  \item Remember, there is no way to “assign to” a variable in ML
    \begin{itemize}
    \item Can only \texttt{shadow} it in a later environment
    \item After binding, a variable’s value is an immutable constant
    \end{itemize}
  \end{itemize}
\end{itemize}

\begin{quote}
\textbf{Try to Avoid Shadowing}
\end{quote}

\begin{verbatim}
val x = "Hello World";
val x = 2;        (* is this a type error? *)
val res = x * 2;  (* is this 4 or a type error? *)
\end{verbatim}

\begin{itemize}
\item Shadowing can be confusing and is often poor style
\item Why? Reintroducing variable bindings in the same
  REPL session may..
  \begin{itemize}
  \item make it seem like \texttt{wrong} code is \texttt{correct}; or
  \item make it seem like \texttt{correct} code is \texttt{wrong}.
  \end{itemize}
\end{itemize}

\begin{quote}
\textbf{Using a Shadowed Variable}
\end{quote}

\begin{itemize}
\item Is it ever possible to use a shadowed variable? \textbf{Yes!}
  \textbf{And no...}
\item It can be possible to uncover a shadowed variable
  when the latest binding goes out of scope
\end{itemize}

\begin{verbatim}
val x = "Hello World";
fun add1(x : int) = x + 1; (* shadow x in func body *)
val y = addl 2;
val z = x ^ "!!"; (* "Hello World!!" *)
\end{verbatim}

\begin{quote}
\textbf{Use \textit{use} Wisely}
\end{quote}

\begin{itemize}
\item Warning: Variable shadowing makes it dangerous to call \texttt{use} more than once without restarting the
  REPL session.
  \begin{itemize}
  \item \texttt{It might} be fine to repeatedly call \texttt{use} in the same REPL session, but unless you know what you’re
    doing, be safe!
  \item Ex: loading multiple distinct files (with independent variable bindings) at the beginning of a
    session
  \item The behavior of \texttt{use} is well-defined, but even expert programmers can get confused
  \item Restart your REPL session before repeated calls to \texttt{use}
  \end{itemize}
\end{itemize}
Debugging Errors

Your mistake could be:

- Syntax: What you wrote means nothing or not the construct you intended
- Type-checking: What you wrote does not type-check
- Evaluation: It runs but produces wrong answer, or an exception, or an infinite loop

Keep these straight when debugging even if sometimes one kind of mistake appears to be another

Play Around

Best way to learn something: Try lots of things and don’t be afraid of errors

Work on developing resilience to mistakes

- Slow down
- Don’t panic
- Read what you wrote very carefully

Maybe watching me make a few mistakes will help...

Boolean Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Type-checking</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>andalso</td>
<td>e1 andalso e2</td>
<td>e1 and e2 must have type bool</td>
<td>Same as Java's e1 &amp;&amp; e2</td>
</tr>
<tr>
<td>orelse</td>
<td>e1 orelse e2</td>
<td>e1 and e2 must have type bool</td>
<td>Same as Java's e1</td>
</tr>
<tr>
<td>not</td>
<td>not e1</td>
<td>e1 must have type bool</td>
<td>Same as Java's !e1</td>
</tr>
</tbody>
</table>

* not is just a pre-defined function, but andalso and orelse must be built-in operations since they cannot be implemented as a function in ML.

• Why? Because andalso and orelse “short-circuit” their evaluation and may not evaluate both e1 and e2.

• Be careful to always use andalso instead of and.

• and is completely different. We will get back to it later.

Style with Booleans

Language does not need andalso, orelse, or not

(* e1 andalso e2 *)
if e1 then e2 else false

(* e1 orelse e2 *)
if e1 then true else e2

(* not e1 *)
if e1 then false else true

(* just say e (!!!) *)
if e then true else false

Comparisons

For comparing int values:

- = < > <= >=

You might see weird error messages because comparators can be used with some other types too:

- > < >= <= can be used with real, but not a mixture of int and real
- * = <> can be used with any ”equality type” but not with real
  - Let’s not discuss equality types yet