Agenda

1. SML Docs
   • Standard Basis

2. First-Class Functions
   • Anonymous
   • Style Points
   • Higher-Order

3. Examples
Standard Basis Documentation

Online Documentation
http://www.standardml.org/Basis/index.html

Helpful Subset
Top-Level http://www.standardml.org/Basis/top-level-chapter.html
List http://www.standardml.org/Basis/list.html
ListPair http://www.standardml.org/Basis/list-pair.html
Real http://www.standardml.org/Basis/real.html
String http://www.standardml.org/Basis/string.html
Anonymous Functions

An Anonymous Function

```
fn pattern => expression
```

- An expression that creates a new function with no name.
- Usually used as an argument to a higher-order function.
- Almost equivalent to the following:

```
let fun name pattern = expression in name end
```

- The difference is that anonymous functions cannot be recursive!!!
Anonymous Functions

What's the difference between the following two bindings?

\[
\begin{align*}
\text{val} & \quad \text{name} = \text{fn} \quad \text{pattern} \Rightarrow \text{expression}; \\
\text{fun} & \quad \text{name} \quad \text{pattern} = \text{expression};
\end{align*}
\]

- Once again, the difference is recursion.
- However, excluding recursion, a \textit{fun} binding could just be syntactic sugar for a \textit{val} binding and an anonymous function.
- This is because there are no recursive \textit{val} bindings in SML.
Unnecessary Function Wrapping

What's the difference between the following two expressions?

\[(\text{fn } xs \Rightarrow \text{tl } xs)\quad \text{vs.}\quad \text{tl}\]

**STYLE POINTS!**

- Other than style, these two expressions result in the exact same thing.
- However, one creates an unnecessary function to wrap \(\text{tl}\).
- This is very similar to this style issue:

\[(\text{if } ex \text{ then true else false})\quad \text{vs.}\quad ex\]
Higher-Order Functions

- A function that returns a function or takes a function as an argument.

Two Canonical Examples

- \texttt{map} : (\texttt{a} \rightarrow \texttt{b}) \times \texttt{a list} \rightarrow \texttt{b list}
  - Applies a function to every element of a list and return a list of the resulting values.
  - \textbf{Example}: \texttt{map (fn x => x*3, [1,2,3]) === [3,6,9]}

- \texttt{filter} : (\texttt{a} \rightarrow \texttt{bool}) \times \texttt{a list} \rightarrow \texttt{a list}
  - Returns the list of elements from the original list that, when a predicate function is applied, result in true.
  - \textbf{Example}: \texttt{filter (fn x => x>2, [~5,3,2,5]) === [3,5]}

\textbf{Note}: List.map and List.filter are similarly defined in SML but use currying. We'll cover these later in the course.
Broader Idea

Functions are Awesome!

• SML functions can be passed around like any other value.
• They can be passed as function arguments, returned, and even stored in data structures or variables.
• Functions like `map` are very pervasive in functional languages.
  – A function like `map` can even be written for other data structures such as trees.
Tree Example

(* Generic Binary Tree Type *)

datatype 'a tree = Empty
           | Node of 'a * 'a tree * 'a tree

(* Apply a function to each element in a tree. *)
val treeMap = fn : ('a -> 'b) * 'a tree -> 'b tree

(* Returns true iff the given predicate returns true when applied to each element in a tree. *)
val treeAll = fn : ('a -> bool) * 'a tree -> bool