

## CSE 341 Section 3

Nick Mooney  
Spring 2017

## Agenda

1. SML Docs
  - Standard Basis
2. First-Class Functions
  - Anonymous
  - Some style tips
  - Higher-Order
3. Examples

## Standard Basis Documentation

### Online Documentation

<http://www.standardml.org/Basis/index.html>

<http://www.smlnj.org/doc/smlnj-lib/Manual/toc.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?

```
val name = fn pattern => expression;
```

```
fun name pattern = expression;
```

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

## Unnecessary Function Wrapping

### What's the difference between the following two expressions?

```
(fn xs => t1 xs)      vs.      t1
```

### Let's look at another example we're familiar with...

```
(if ex then true else false) vs. ex
```

- Other than style, these two expressions result in the exact same thing.
- However, one creates an unnecessary function to wrap `t1`.
- Style points, blah blah... do it because it's nice, not for the points 😊

# Higher-Order Functions

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

## Two Canonical Examples

- `map : ('a -> 'b) * 'a list -> 'b list`
  - Applies a function to every element of a list and return a list of the resulting values.
  - Example: `map (fn x => x*3, [1,2,3]) === [3,6,9]`
- `filter : ('a -> bool) * 'a list -> 'a list`
  - Returns the list of elements from the original list that, when a predicate function is applied, result in true.
  - Example: `filter (fn x => x>2, [~5,3,2,5]) === [3,5]`

**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.