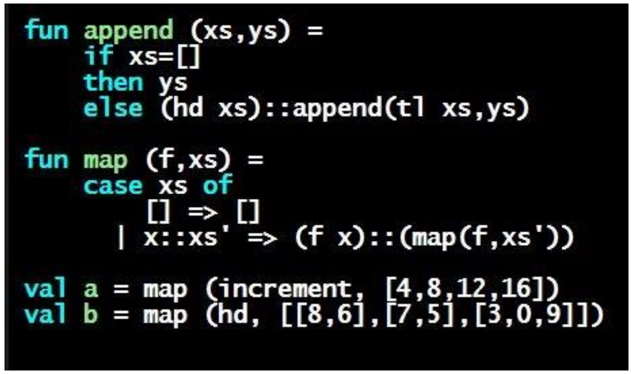

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

fun map (f,xs) =
    case xs of
        [] => []
      | x::xs' => (f x)::(map(f,xs'))

val a = map (increment, [4,8,12,16])
val b = map (hd, [[8,6],[7,5],[3,0,9]])
```

Programming Languages

Dan Grossman

Mixins

Mixins

- A *mixin* is (just) a collection of methods
  - Less than a class: no instances of it

- Languages with mixins (e.g., Ruby modules) typically let a class have one superclass but *include* number of mixins

- Semantics: *Including a mixin makes its methods part of the class*
  - Extending or overriding in the order mixins are included in the class definition
  - More powerful than helper methods because mixin methods can access methods (and instance variables) on **`self`** not defined in the mixin

Example

```
module Doubler
  def double
    self + self # assume included in classes w/ +
  end
end
class String
  include Doubler
end
class AnotherPt
  attr_accessor :x, :y
  include Doubler
  def + other
    ans = AnotherPt.new
    ans.x = self.x + other.x
    ans.y = self.y + other.y
    ans
end
```

Lookup rules

Mixins change our lookup rules slightly:

- When looking for receiver **`obj`**'s method **`m`**, look in **`obj`**'s class, then mixins that class includes (later includes shadow), then **`obj`**'s superclass, then the superclass' mixins, etc.

- As for instance variables, the mixin methods are included in the same object
  - So usually bad style for mixin methods to use instance variables since a name clash would be like our **`CowboyArtist`** pocket problem (but sometimes unavoidable?)

The two big ones

The two most popular/useful mixins in Ruby:

- Comparable: Defines **`<`**, **`>`**, **`==`**, **`!=`**, **`>=`**, **`<=`** in terms of **`<=>`**
- Enumerable: Defines many iterators (e.g., **`map`**, **`find`**) in terms of **`each`**

Great examples of using mixins:

- Classes including them get a bunch of methods for just a little work
- Classes do not "spend" their "one superclass" for this
- Do not need the complexity of multiple inheritance

- See the code for some examples

Replacement for multiple inheritance?

- A mixin works pretty well for **`ColorPt3D`**:
  - Color a reasonable mixin except for using an instance variable

```
module Color
 attr_accessor :color
end
```

- A mixin works awkwardly-at-best for **`ArtistCowboy`**:
  - Natural for **`Artist`** and **`Cowboy`** to be **`Person`** subclasses
  - Could move methods of one to a mixin, but it is odd style and still does not get you two pockets

```
module ArtistM …
class Artist < Person
   include ArtistM
class ArtistCowboy < Cowboy
   include ArtistM
```