```
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
2013

Visibility

Who can access what

- We know “hiding things” is essential for modularity and abstraction
- OOP languages generally have various ways to hide (or not) instance variables, methods, classes, etc.
  - Ruby is no exception
- Some basic Ruby rules here as an example…

Object state is private

- In Ruby, object state is always private
  - Only an object's methods can access its instance variables
  - Not even another instance of the same class
  - So can write `@foo`, but not `e.@foo`

- To make object-state publicly visible, define "getters" / "setters"
  - Better/shorter style coming next

```
def get_foo
  @foo
end
def set_foo x
  @foo = x
end
```

Conventions and sugar

- Actually, for field `@foo` the convention is to name the methods

```
def foo
  @foo
end
```

```
def foo= x
  @foo = x
end
```

- Cute sugar: When *using* a method ending in `=`, can have space before the `=`

```
e.foo = 42
```

- Because defining getters/setters is so common, there is shorthand for it in class definitions
  - Define just getters: `attr_reader :foo, :bar, …`
  - Define getters and setters: `attr_accessor :foo, :bar, …`

- Despite sugar: getters/setters are just methods

Why private object state

- This is “more OOP” than public instance variables
- Can later change class implementation without changing clients
  - Like we did with ML modules that hid representation
  - And like we will soon do with subclasses
- Can have methods that “seem like” setters even if they are not

```
def celsius_temp= x
  @kelvin_temp = x + 273.15
end
```

- Can have an unrelated class that implements the same methods and use it with same clients
  - See later discussion of “duck typing”

Method visibility

- Three *visibilities* for methods in Ruby:
  - **`private`**: only available to object itself
  - **`protected`**: available only to code in the class or subclasses
  - **`public`**: available to all code

- Methods are **`public`** by default
  - Multiple ways to change a method's visibility
  - Here is one way…

Method visibilities

```
class Foo =
# by default methods public
   …
 protected
# now methods will be protected until
# next visibility keyword
   …
 public
   …

 private
   …
end
```

One detail

If `m` is private, then you can only call it via `m` or `m(args)`

- As usual, this is shorthand for `self.m` …
- But for private methods, only the shorthand is allowed