```
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

Duck Typing

Duck Typing

"If it walks like a duck and quacks like a duck, it's a duck"

- Or don't worry that it may not be a duck

When writing a method you might think, "I need a `Foo` argument" but really you need an object with enough methods similar to `Foo`'s methods that your method works

- Embracing duck typing is always making method calls rather than assuming/testing the class of arguments

Plus: More code reuse; very OOP approach

- What messages an object receive is "all that matters"

Minus: Almost nothing is equivalent

- `x+x` versus `x*2` versus `2*x`
- Callers may assume a lot about how callees are implemented

Duck Typing Example

```
def mirror_update pt
  pt.x = pt.x * (-1)
end
```

- Natural thought: "Takes a **`Point`** object (definition not shown here), negates the **`x`** value"
  - Makes sense, though a **`Point`** instance method more OOP
- Closer: "Takes anything with getter and setter methods for **`@x`** instance variable and multiplies the **`x`** field by **`-1`**"
- Closer: "Takes anything with methods **`x=`** and **`x`** and calls **`x=`** with the result of multiplying result of **`x`** and **`-1`**"
- Duck typing: "Takes anything with method **`x=`** and **`x`** where result of **`x`** has a **`*`** method that can take **`-1`**. Sends result of calling **`x`** the **`*`** message with **`-1`** and sends that result to **`x=`**"

With our example

```
def mirror_update pt
  pt.x = pt.x * (-1)
end
```

- Plus: Maybe **`mirror_update`** is useful for classes we did not anticipate
- Minus: If someone does use (abuse?) duck typing here, then we cannot change the implementation of **`mirror_update`**
  - For example, to **`- pt.x`**
- Better (?) example: Can pass this method a number, a string, or a **`MyRational`**

```
def double x
  x + x
end
```