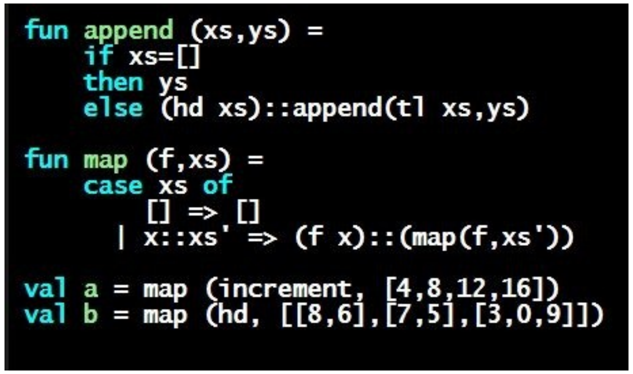

```
fun append (xs,ys) =
    if xs=[]
    then ys
    else (hd xs)::append(tl xs,ys)

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

Everything is an Object

Pure OOP

- Ruby is fully committed to OOP:
  Every value is a reference to an object

- Simpler, smaller semantics

- Can call methods on anything
  - May just get a dynamic “undefined method” error

- Almost everything is a method call
  - Example: **`3 + 4`**

Some examples

- Numbers have methods like `+`, `abs`, `nonzero?`, etc.
- `nil` is an object used as a “nothing” object
  - Like `null` in Java/C#/C++ except it is an object
  - Every object has a `nil?` method, where `nil` returns `true` for it
  - Note: `nil` and `false` are “false”, everything else is “true”
- Strings also have a `+` method
  - String concatenation
  - Example: `"hello" + 3.to_s`

All code is methods

- All methods you define are part of a class
- Top-level methods (in file or REPL) just added to **`Object`** class
- Subclassing discussion coming later, but:
  - Since all classes you define are *subclasses* of **`Object`**, all *inherit* the top-level methods
  - So you can call these methods anywhere in the program
  - Unless a class overrides (*roughly-not-exactly*, shadows) it by defining a method with the same name

Reflection and exploratory programming

- All objects also have methods like:
  - **`methods`**
  - **`class`**

- Can use at run-time to query "what an object can do" and respond accordingly
  - Called *reflection*

- Also useful in the REPL to explore what methods are available
  - May be quicker than consulting full documentation

- Another example of "just objects and method calls"