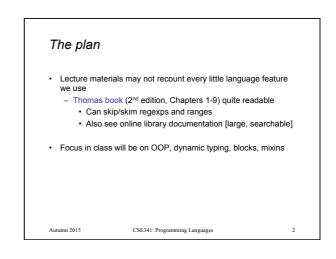
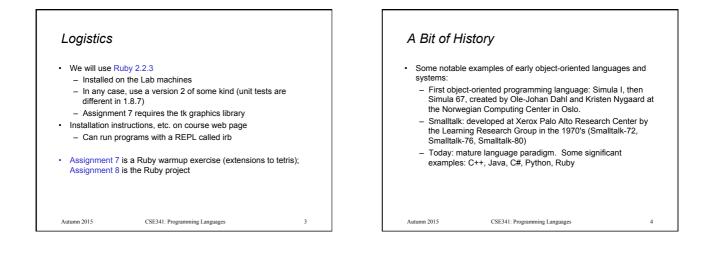


Alan Borning Autumn 2015 (slides borrowed from Dan Grossman)

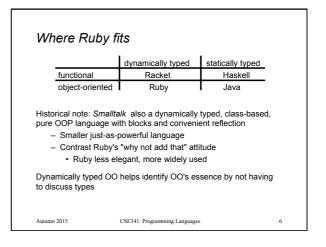


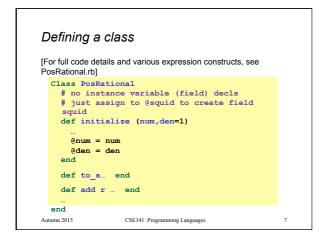


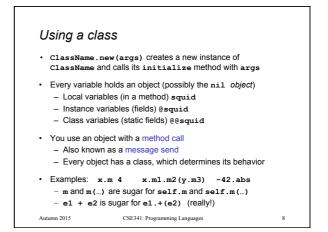
Ruby

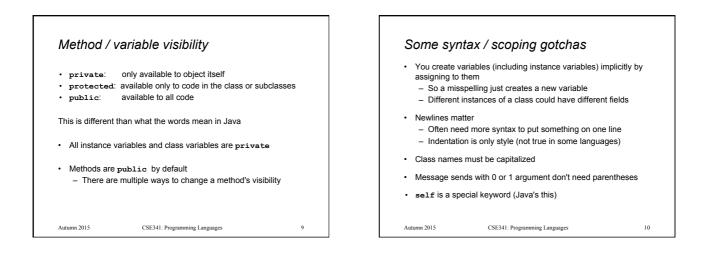
- · Pure object-oriented: all values are objects (even numbers)
- Class-based: Every object has a class that determines behavior - Like Java, unlike Javascript
 - Mixins (neither Java interfaces nor C++ multiple inheritance)
- Dynamically typed
- Convenient reflection: Run-time inspection of objects
- Blocks and libraries encourage lots of closure idioms
- Syntax and scoping rules of a "scripting language"
 - Often many ways to say the same thing
 - Variables "spring to life" on use
 - Lots of support for string manipulation [we won't do this]
- · Popular for building server-side web applications (Ruby on Rails) CSE341: Programming Languages

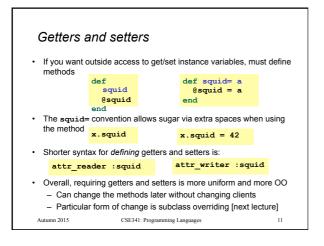
Autumn 2015

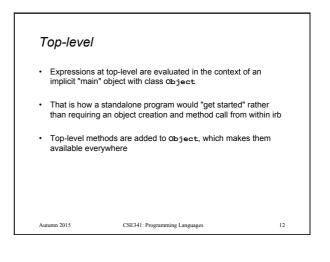












Class definitions are dynamic

- · All definitions in Ruby are dynamic
- Example: Any code can add or remove methods on existing classes
 - Very occasionally useful (or cute) to add your own method to the Array class for example, but it is visible to all arrays
- · Changing a class affects even already-created instances
- Disastrous example: Changing Fixnum's + method
- Overall: A simple language definition where everything can be changed and method lookup uses instance's classes

13

Autumn 2015 CSE341: Programming Languages

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 toad argument" but really you need an object with enough methods similar to toad'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 OO 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

