

Using Arrays

- · See many examples, some demonstrated here
- Consult the documentation/tutorials
 If seems sensible and general, probably a method for it
- · Arrays make good tuples, lists, stacks, queues, sets, ...
- Iterating over arrays typically done with methods taking blocks

 Next topic...

Autumn 2019 CSE341: Programming Languages

11aors 4

Blocks

Blocks are probably Ruby's strangest feature compared to other PLs

But almost just closures

- Normal: easy way to pass anonymous functions to methods for all the usual reasons
- Normal: Blocks can take 0 or more arguments
- Normal: Blocks use lexical scope: block body uses environment where block was defined

Examples:

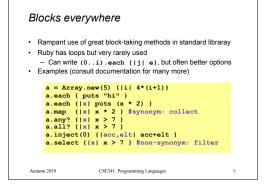
3.times { puts "hi" } [4,6,8].each { puts "hi" } i = 7 [4,6,8].each { |x| if i > x then puts (x+1) end } Autumn 2019 CSE341: Pogramming Languages 5

Some strange things

- Can pass 0 or 1 block with any message
- Callee might ignore it
- Callee might give an error if you do not send one
- Callee might do different things if you do/don't send one
 Also number-of-block-arguments can matter
- Just put the block "next to" the "other" arguments (if any)
 Syntax: {e}, {|x| e}, {|x,y| e}, etc. (plus variations)
 - Can also replace { and } with do and end
 Often preferred for blocks > 1 line

6

Autumn 2019 CSE341: Programming Languages





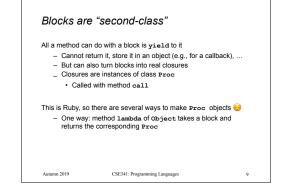
- Callee does not give a name to the (potential) block argument
- Instead, just calls it with yield or yield(args)

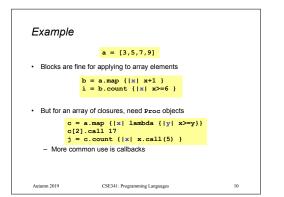
 Silly example:
 def silly a
 (yield a) + (yield 42)
- end – See code for slightly less silly example
- Can ask block_given? but often just assume a block is given or that a block's presence is implied by other arguments

8

11

Autumn 2019 CSE341: Programming Languages

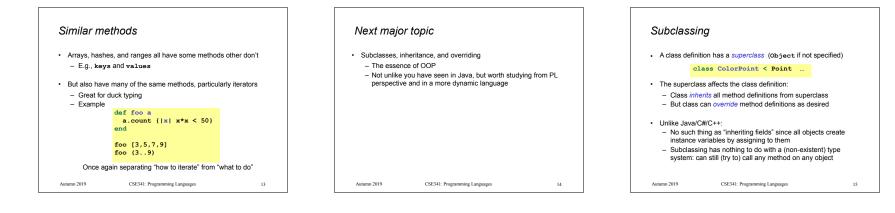


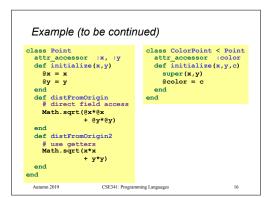


Moral

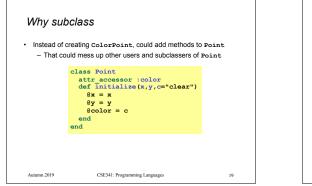
- First-class ("can be passed/stored anywhere") makes closures more powerful than blocks
- · But blocks are (a little) more convenient and cover most uses
- This helps us understand what first-class means
- Language design question: When is convenience worth making something less general and powerful?

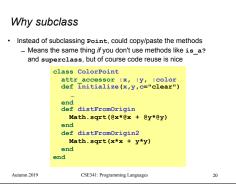
Autumn 2019 CSE341: Programming Languages

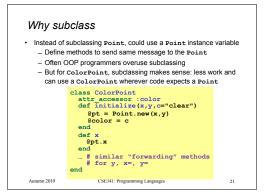


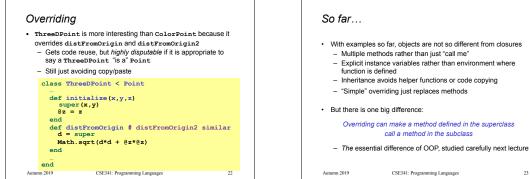


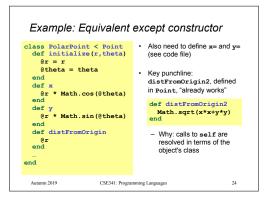
Example	continued	
Consider alt	ernatives to:	
	<pre>class ColorPoint < Point attr_accessor :color def initialize(x,y,c) super(x,y) @color = c end end</pre>	
	ssing is a good choice, but program in OOP languages	nmers often overuse
Autumn 2019	CSE341: Programming Languages	18











- · With examples so far, objects are not so different from closures
- Multiple methods rather than just "call me"

Overriding can make a method defined in the superclass call a method in the subclass

23