Slide 1

Administrivia

- (None?)

Slide 2

Arrays and Lists — Overview

- With what we've done so far we have enough tools to compute anything we want to compute. (And in combination with input/output redirection we could probably do some interesting things.)
- However, some things are awkward (repetition), and we don't yet have a convenient way to store many values — something similar to subscripted values in math.
- Most programming languages give you a way to represent *collections*. Exactly what you get depends on the language — e.g., C gives you only something quite primitive (but close to what the hardware can do), Java gives you something more abstract/useful, and Scala goes even further.

Slide 3

Arrays and Lists in Scala

- Scala provides two ways of representing a "sequence" (ordered list of elements), `Array`s and `List`s. From the outside they look very similar, but behave fairly differently:

  An `Array` has a fixed number of elements, but the values of the elements can be changed.

  A `List` cannot be changed at all, but there are easy and efficient ways to build lists.

- Both are "parameterized types", which means you can specify the type of the elements.

Slide 4

Arrays in Scala

- Two syntaxes for creating an `Array`. Examples:

```
// four elements, initial values as given
val a1 = Array(1,2,3,4)
// ten elements, all zero
val a2 = new Array[Int](10)
```

- Syntax for referencing element uses name of array plus index in parentheses. Indexes range from 0 through length minus 1. Examples:

```
println(a1(1))
a2(2) = 20
```

Slide 5

Arrays in Scala, Continued

- Length of `Array` can be obtained with `.length` or `.size`.
- That gives us enough to write some simple functions using recursion . . .

Slide 6

Recursion — Review/Recap

- A function (or definition) is recursive if it calls/uses itself. Obviously(?) there needs to be at least one base case too.
- Can be somewhat tricky to think about whether/how recursive functions work — it involves nested calls to the same function, one "inside" the other in some sense. May be helpful to take what I call a "static" perspective, focusing on the code and one call to the function rather than the whole bunch of nested calls.
- To do that, first be clear on what the function does — "computes n factorial", or "computes the sum of array elements starting at this index". Then ask . . .

Slide 7

Recursive Functions — "(How) Does it Work"?

- (How) does it work for the base case(s)?
- (How) does it work for the non-base cases, *assuming that the recursive calls work*, meaning that they do what the function is supposed to do, based on the definition you came up with.
- (How) does each recursive call get us closer to a base case?
- (In some ways this is a mirror image of induction, as in proofs by induction, where we start with small cases and construct more complex ones.)

Slide 8

Lists in Scala

- As with arrays, there are two basic ways to make lists in Scala.
- One is similar to how you create an array by listing elements:

```
val l1 = List(1,2,3,4)
```

- Another is to build it up an element at a time with the "cons" operator (`::`):

```
var ll = List[Int]()
ll = 1::ll
```

(You would likely not write exactly that code; it's meant only to illustrate use of the `::` operator.)

Slide 9

Lists in Scala, Continued

- You *can* get the length of a list and values of elements using the same syntax as with arrays, but that's not very idiomatic.
- Better is to use to use `head` and `tail` and recursion. Let's write something analogous to the array demo . . .

Slide 10

Arrays and Lists and Recursion, More Examples

- We started with functions to read numbers into an array or a list and print them out. What else can we do with them? Lots of things . . .
- We could write functions that take a collection of numbers and return a single number. Examples include sum, product, max, min, . . .
- But all of these functions basically do the same thing, right? the only thing that's different is how we combine two numbers into one. So maybe what we really want is a function one of whose parameters is a function . . . (To be continued!)

Slide 11

Minute Essay

- Anything today particularly unclear?