Slide 1

Administrivia

- Homework 5 on the Web; due in a week. Next quiz will also be next week.

Slide 2

Minute Essay From Last Lecture

- One person says she understands all the parts of the example programs but can't always put them together herself, and do I remember any tips that helped me . . .

  I wish I did! I do remember some light-bulb moments but not what triggered them.

- Another asks whether examples will be online. Usually yes! but also often takes a day or so.

Slide 3

Arrays and Lists — Review/Recap

- Scala provides two basic types of "sequences", arrays and lists.
- Several ways to work with them. We start out by applying tools we already have (recursive functions), partly to get more practice with them. Also an opportunity to revisit "higher-order functions" (functions that use other functions as parameters) . . .

Slide 4

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 . . .

Slide 5

Higher-Order Functions — Review/Recap

- "Higher-order functions" (first discussed in chapter 5) are functions that use other functions as parameters (or as return values). Very useful concept, supported in fairly different ways in different languages.
- As an example of how this is useful — summing all elements of an array versus computing their product, versus finding the smallest or largest element, etc. Basic computation (a *reduction*) involves combining elements pairwise with a binary operator, and by using a higher-order functions we don't have to repeat the parts that are the same.

Slide 6

Defining Higher-Order Functions in Scala

- Syntax illustrated by an example for our demo program(s):

```
def arrayCombine(a : Array[Int], startIndex : Int,
  combine : (Int, Int) => Int, identity : Int) : Int = { /* .... */ }
```

  where `combine` is a parameter that is itself a function(!).

  (I could have put all of that on one line, but it would have been long.)

- Within the body of the function (`arrayCombine` in the example) we can call the parameter function (`combine`) as we usually do, e.g., `combine(1, 2)` to call the function with parameters 1 and 2.

Slide 7

Using Higher-Order Functions in Scala

- One option for function parameters is a named function:

```
def add(x : Int, y : Int) : Int = { x + y }
arrayCombine(a, 0, add, 0)
```

- Another option is a function literal:

```
arrayCombine(a, 0, (x, y) => ( x + y ), 0)
```

- Yet another option is a special form of a function literal:

```
arrayCombine(a, 0, _ + _, 0)
```

Slide 8

Example(s) Revisited

- We could now revise our array demo program to do still more things with the array — find minimum and maximum elements, for example.
- We could add similar functionality to our list demo program.

Slide 9

Collection Methods — Overview

- As noted earlier, both arrays and lists provide a wide range of interesting(?) methods. (“Methods”? Briefly, special type of functions, described a bit in chapter 3.) The textbook lists some of them and is a good starting point. For full details, however . . .

Slide 10

The Scala API

- In context, API means “Application Programming Interface”. Meant as complete documentation of the language’s library functions, methods, etc. Many languages and libraries have one of these.
- The standard presentation of Scala’s API is descended from Java and is nicely organized for online browsing (link from course “Useful links” page). Worthwhile spending a bit of time learning how to find things in it (though not everything will make sense yet).

Slide 11

The Scala API — Tips/Gotchas

- Notice — some entries in left frame show two icons ("o" and "c"). "c" shows things you can do with objects of whatever type it is (e.g., `Int`s). "o" shows things you can do with `Int` itself — e.g., get minimum and maximum value.
- Some things are documented in unobvious places (e.g., `ArrayOps`, `StringOps`, `RichInt`).

Slide 12

Collection Methods — Basics

- Some methods to extract parts of a collection:
  `drop`, `init`, `last`, `slice`, `splitAt`, `take`, `takeRight`
- Some methods to test something about a collection:
  `contains`, `endsWith`, `isEmpty`, `nonEmpty`, `startsWith`
  `indexOf`, `lastIndexOf`
- Some other useful methods and variables:
  `foreach`, `mkString`, `reverse`, `zip`, `zipWithIndex`, `length`, `size`

Slide 13

Collection Methods — Basics Continued

- `sum` and `product` work on objects that support addition and multiplication.
- `min` and `max` work on objects that can be put in order.
- Strings have `split`.

Slide 14

Collection Methods — Higher-Order Methods

- `exists`, `forall`
- `filter`, `partition`
- `map`
- `reduceLeft`, `foldLeft`

Slide 15

Examples

- Right away we have alternatives to most of the functions in our "demo" program. (But that's okay — they were good practice.)
- A somewhat more interesting example: Find out whether a line of text is a palindrome. Simplest version is, well, simple with `reverse`. If we want to implement the usual definition, though, that looks only at letters and ignores case?

Slide 16

Minute Essay

- Can you think of other interesting things you could do with some of these methods?