Slide 1

Administrivia

- Homework 5 on the Web; due next Tuesday.

Slide 2

Minute Essay From Last Lecture

- Minute essay question from before the midterm: How much time outside class do you spend on this course?
- Answers varied, but many less than general-rule “one credit hour means three hours total, in and out of class”.
- A few people mentioned spending time just trying things in the REPL. A good way to learn!

Arrays, Lists, and Recursion — Review/Recap

Slide 3

- Recall from before midterm — arrays and lists, and example programs using recursion to work on them.
- Just this much gives us the ability to do things we couldn't before. But both types of collections also provide a wide range of interesting(?) methods ("collection methods"). Before we dive into those, however . . .

Higher-Order Functions — Review/Recap

Slide 4

- "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 5

Defining Higher-Order Functions in Scala

- Syntax illustrated by our example from class:

```
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 6

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)
```

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 7

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 8

Slide 9

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 10

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 11

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 12

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 13

Collection Methods — Higher-Order Methods

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

Slide 14

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 15

Minute Essay

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