Slide 1

Administrivia

- Reminder: Homework 5 due today.

Slide 2

Minute Essay From Last Lecture

- Something to keep track of expenses?
- General sorting of data.
- Tests of statistical significance, e.g., correlations. Probability problems.
- Games (video and otherwise).
- Solving Sudoku problems. Seems hard. Tic-tac-toe might be easier? (Or not — may be less computation but more thinking to write program.)
- Graphing/plotting, like a graphing calculator.
- Grades program that would also tell you what you need to get on future assignments.

Slide 3

Minute Essay From Last Lecture, Continued

- Sums of geometric and arithmetic sequences, other math.
- Program to recommend products, like what online retailers use. (Hm. How would it decide?)
- Program that would check courses you've taken against courses you need and shows what you still need. (Maybe interface with TigerPAWS?)
- Chess program.
- Something for a company that would rank customer e-mail by importance. (How would it decide?)

Slide 4

Collection Methods — Summary

- Scala offers many, many ways to operate on elements of a collection. Programs that use them are apt to be compact but not necessarily easy to understand right away.
- Tip: If you can't understand what a complicated combination of these methods does, try executing pieces of it in the REPL. (This is also useful when writing programs — build up a complicated expression a little at a time in the REPL, then copy it into your program.)

Slide 5

One More Construct — Loops

- With what we have already we can build programs to do pretty much anything programs can do — but sometimes in a way that seems a little forced.
- Like arrays and lists, loops give you one more thing in your "bag of tricks" and can sometimes produce code that's simpler and easier to understand. Functional languages tend to use recursion to achieve iteration; imperative languages tend to use loops.

Slide 6

`while` and `do while` Loops in Scala

- These loops repeat a statement or block (the *loop body*) while some condition (the *loop condition*) is true. One variant (`while`) tests the loop condition before each repetition; the other (`do while`) tests after each repetition. Normally the loop body contains something that moves to the next iteration.
- Simple example (prints values 0 through 9):

```
var n = 0
while (n < 10) {
  println(n)
  n += 1
}
```

- (Most languages that support loops offer something that looks pretty similar to this.)

Slide 7

`while` and `do while`, Continued

- These two forms of loops are particularly useful when you don't know in advance how many times you need to execute the loop body. (So, the simple example is not really a good one.)
- However, they have some drawbacks — such as what happens when you forget to put something in the loop body that moves to the next iteration. So . . .

Slide 8

`for` Loops in Scala

- These loops let you repeat a statement or block (the loop body) for a sequence of values. Most languages that support loops offer something along similar lines, but it may be significantly less capable.
- Simple examples similar to what most languages support:

```
for (i <- 0 to 9) {
  println(i)
}
for (i <- 0 until 10) {
  println(i)
}
```

`for`, Continued

- In Scala, however, what comes after the `for` and `<-`) is actually a form of sequence. (Try typing `0 to 9` in the REPL.)
- So we can also write `for` loops that operate on all values of a collection, for example.
- Multiple generators (see the textbook) allows us to concisely do what might require nested loops in another language.

Examples

- Simple example — summing integers read from standard input, ending with “quit”.
- Another simple example — array/list demos revised to use loops.
- More complex example — finding primes less than some limit value using the “sieve of Eratosthenes” approach.

Minute Essay

- None — quiz.