Slide 1

Administrivia

- FYI: Our student ACM ("Association for Computing Machinery", major professional association) chapter is doing tutoring, Mondays through Thursdays from 3:30pm to 5pm in HAS 329. Another option for getting help with homework!
- Reminder: Midterm Tuesday. Review sheet on the Web. Also quiz solutions and sample solutions to homeworks.
- I will have office hours tomorrow afternoon (more info by e-mail).

Slide 2

About the Midterm

- Review class notes, homeworks, readings. If I didn't mention it in class, odds are I won't ask about it on the exam.
- Questions will be a mix of problems similar to those in quizzes, plus possibly some true/false, multiple choice, or short-answer.
- Open book, open notes. Okay to use a computer to review book, your notes and graded work, and the course Web site, but nothing else. No using the Scala interpreter/compiler to answer questions about programs, sorry.
- (Topic by topic through the review sheet.)

Slide 3

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 4

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 5

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 6

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 7

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 from last time . . .

Slide 8

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 9

Minute Essay

- About how much time a week are you spending on this class outside of class? How much of it involves actually programming, or at least trying things in the REPL? (Keep in mind the minute-essay mantra — “no wrong answers”.)