Slide 1

Administrivia

- Reminder: Homework 3 due Thursday. Quiz 2 Thursday as well.
- Midterm scheduled for next Tuesday. Postpone?

Slide 2

Recursion — Review/Recap

- Key ideas in recursion:
  - One or more base cases that can be solved without recursion.
  - A way of splitting up other cases into one or more *smaller* recursive calls plus some other logic.
- Very important that recursive calls be somehow smaller, so that you eventually reach a base case!
- Simple examples often based on math functions defined recursively (factorial, Fibonacci numbers).
- Simple examples — program to "count down", program to compute sum.
- Another example — four-function calculator. (Implement in a way that illustrates use of `match`, functions as objects.)

Slide 3

Sidebar: Tracing Code

- Something I sometimes ask you to do on quizzes/exams is tell me what a program prints out (without just typing it in). To do this you need to “trace the program”, “play computer”, etc.
- (This can be a useful skill when your programs don’t give the answers you want.)
- To do this: Work through the program statements in the same order the computer does, writing down values for variables.

Slide 4

Sidebar: Testing Programs

- As you’re learning, the first step in writing a working program is just coming up with something the compiler-or-whatever can understand.
- But once you have that you may still have logic errors (even if — as you should! — you’ve thought some about whether your approach should work), and if your logic is okay you might have mistyped something.
- So before “shipping” it, good to test . . .
- With what inputs? Choosing good test inputs maybe has aspects of both art (craft?) and science, but some thoughts . . .

Slide 5

Sidebar: Testing Programs, Continued

- Test with inputs that, among them, cover all the paths through your code.
- Test with "boundary values" if appropriate.
- If possible, test with inputs such that you can easily confirm that the answer is right!
- And you may find it helpful to recall . . .

Slide 6

Arrays and Lists — Preview

- With what we've done so far we have enough tools to compute anything we want to compute.
- 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.

Minute Essay

- Would you be okay with postponing the midterm until after the (short) fall break? i.e., until October 21 (T) or October 23 (Th)?