Administrivia

Slide 1

- (Review minute essay from last time.) Notice that when there's an answer it will be in the not-preliminary version of the slides/notes online.
- I say in the syllabus that I try to respond promptly to e-mail. Exceptions are minute essays and homeworks, which I don't always look at right away. If you need a quick reply, make that apparent on the subject line please!

More Administrivia

Slide 2

- Homework 2 to be on the Web soon. I will send mail. Due in a week. Homework 1 grades/comments to be sent by e-mail.
- If you need help with homework — and you may! — you can ask me (office hours or e-mail), or the ACM student chapter will probably be offering peer tutoring.

Slide 3

Scala and Representing Data — Review/Recap

- All data in Scala (and many/most other programming languages) has a "type" (that among other things defines a set of possible values and operations on those values).
- Numeric types include `Int`, `Long`, `Float`, `Double`. Operations include familiar(?) arithmetic operators.
- Text types include `Char`, `String`. Operations on `String` include `+` defined to mean string concatenation.

Slide 4

Variables

- We know enough — more than enough — at this point to use the Scala REPL as a calculator. But that's not really programming, since if we want to do the same calculation for different sets of values we'd have to retype everything.
- To do almost anything interesting, we need some way to save values and give them names, so we can reference them again. So Scala, like most programming languages, has a notion of *variables*, similar (but not identical!) to variables in math. (The biggest difference is that some Scala variables can take on different values as a calculation proceeds.)
- Basic syntax for defining variables requires a keyword (`val` or `var`), a type, a name, and a value. Can omit type if Scala can guess. `val` versus `var`? Former can't change value, latter can (with *assignment statement*, almost identical to definition but without `var`). Value is expressed as an expression, which can mention other previously-defined variables and which at runtime is *evaluated* to give a value.

Slide 5

Getting Input

- We need one more thing in order to write real (if very small!) programs — a way to get input from the human user of the program.
- In Scala, one way is to use library functions `readInt`, `readDouble`, etc., (`readLine` for strings), e.g.

  ```
  val input = readInt
  ```

  (Caveat: In newest version of Scala — installed on some machines — this gives a warning about deprecated function. Use `import io.StdIn._` to avoid.)

  Notice what happens if you type in something other than a number. (No, it's not very pretty, but for now it will do, and we will talk later about alternatives.)

Slide 6

What We Know How To Do — Review

- Write expressions including numeric and character-data literals.
- Define variables and give them values.
- "Print" things (display them on standard output, in techiespeak). (How do we print values of variables?)
- Get input from standard input ("the keyboard" for now) with `readInt`, `readDouble`, `readLine`.

Slide 7

Example

- As a first example, write a program that “counts out change” — for a given number of cents, says how many dollars, quarters, etc., are needed.
- First step — understand the problem. Often helpful to work through some examples “by hand”.
- Next, figure out how to get the same result(s) by using things in your “bag of tricks” (right now pretty limited, but will grow as you learn more).
- Programming tip: Can be helpful to try things out (e.g., ways of doing calculation) in REPL. Collect for reuse in `.scala` file (“Scala program” or, for the pedantic, “Scala script”).

Slide 8

Minute Essay

- Anything today unclear?