Slide 1

Administrivia

- Reminder: Quiz 1 Thursday. Questions based on reading and lectures so far (through today). Likely to focus on material in chapter 3. “Open book/notes”, meaning access to textbook, your notes, anything on course Web site.
- Homework 2 on the Web. Due a week from today. (If you have not turned in Homework 1 — please do. Ask if questions.)
- (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.)
- (Also, in minute essays you can ask any questions that occur to you about the class or related subjects, and I’ll try to answer.)

Slide 2

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 3

Objects and Methods — Review

- Text strings don't really correspond to anything the hardware can work with as directly as it works with integer and floating-point numbers. So how to represent them is left somewhat more to the discretion of the programming language. They're a simple example of a kind of thing we might want to be able to work with that's somewhat more complicated than what the hardware provides.
- To make working with things other than simple numbers easy, Scala, again like many (but not all!) programming languages has a notion of *objects* (i.e., it is an *object-oriented* (OO) language).
- Remember that we defined a type as a set of values together with some operations on them? In OO-speak, an *object* is something with a value of a particular type, and its *methods* are operations that the type says can be done on it (e.g., arithmetic operations on integers).

Slide 4

Objects and Methods in Scala

- In Scala (unlike some other popular programming languages), everything is an object. This makes some things very convenient (though it puts a certain distance between the language and the hardware, which *may* have negative effects on performance).
- Some operations on objects just do something, without any need for more information (e.g., `toInt` converts a `Double` to an `Int`). Others requre *parameters* (e.g., integer addition).
- Basic syntax for invoking an object's methods requires a period, the name of the method, parentheses, and any parameters. Scala allows many of these to be omitted if it can figure out what you mean. (Indeed, some methods that take no parameters must *not* be followed by parentheses — e.g., `toDouble`.)

Slide 5

Objects and Methods in Scala

- Many useful "library" methods built into the language. The REPL provides some support in the form of tab completion. (Try some things with integers and strings!)
- Library methods include many for working with text strings, plus the `math` object. See book for details.

Slide 6

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 7

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`.

Slide 8

Example

- As a first example, write a program that "makes 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. (Don't skip this!)
- Next, figure out how to get the same result(s) by using things in your "bag of tricks".

Slide 9

Minute Essay

- Anything today unclear?