Opening Discussion

- Midterm answers.
- Minute essay comments
  - Arrays if matrices.
  - Using a HUD while driving.
  - Advice on low cost gaming machine.
  - How fast can I skate 20 laps and do I play roller hockey?
  - Consider watching instead of following.
Recursion is sufficient for making repetition, but in imperative languages it isn't the normal approach. Instead, people use loops.

The simplest loop is the while loop.

- while(condition) statement

The condition is evaluated first. If it is true the statement (possibly a block) executes.

This repeats until the condition is false.
The partner to the while loop is the do-while loop.

- do {
  - statement
  - } while(condition)

- This loop is post-check instead of the pre-check of the normal while loop.

- Always happens once.

- The while loop might never happen.
The most commonly used loop in most languages is the for loop. The Scala version is a bit different from most.

Often used for counting:
- `for(i <- 1 to 10) { ... }`

In general it is a “for each” loop that goes through a collection.
- `for(e <- coll) { ... }`

Variable takes on value of each element in the collection.
Range Type

- Range types provide an easy way to make collections for counting.
- “to” and “until” operate on numeric types to produce ranges.
  - 1 to 10
  - 0 until 10
- Use “by” to change the stepping in a range.
  - 1 to 100 by 2
  - 10 to 1 by -1
  - 'a' to 'z' by 3
The for loop can be used as an expression if you put yield between the end of the for and the expression after it.

- `for(e <- coll) yield expr`

What you get back will be a collection that is generally of the same type as what you iterated over.
if Guards

- You can put conditions in the for that will cause some values to be skipped.
  - for(n <- nums; if n%2==0) ...


Multiple Generators

- You can also put multiple generators in a for loop.
  - for(i <- 1 to 10; j <- i to 10) ... 
- You can combine as many generators and guards as you want. You can also declare variables in the middle of the for.
- The thing you assign into is like a val so it can be a “pattern”. We have only seen this with tuples so far.
You can have collections of collections. A common example would be something like `Array[Array[Double]]` to represent a matrix.

Both fill and tabulate can be used to make these.

```scala
val ident=Array.tabulate(3,3)((i,j) => if(i==j) 1.0 else 0.0)
```
Motivation

- Programs are more useful when they can interact with files.
- Everything that isn't in a file is lost when the program stops running.
I/O Redirection

- Using I/O redirection gives you some very basic ability to read from and write to files.
- It has big limitations though because there is only one file each way.
- More over, that one file blocks the ability to use either standard input or output.
To read from a file we will be using the scala.io.Source type. To understand what that means, we need to talk about packages.

Packages provide a way to organize code and group things of like functionality.

Import statements let you use things without typing in their fully specified names.
To get a sense of the different package in Scala, it is helpful to look at the API.

There are still lots of things in the API you won't fully understand. That isn't a problem as you aren't expected to get too much from it right now.
Call `Source.fromFile(fileName: String)` to get a `Source` object that reads from a file.

There are other methods in the main `Source` object that we will learn about later.

The `fromFile` method technically gives you `BufferedSource`. This is for efficiency.
Iterators

- Both Source and BufferedSource are of the type Iterator[Char].
- An Iterator has most of the methods you are used to from List and Array. However, you can only go through it once.
- Fundamentally uses hasNext and next methods.
getLines

- This will give you an Iterator[String] that will go through the file one line at a time instead of a character at a time.
- You will often find this more useful.
What questions do you have?

IcP #5 on Friday (note this is moving back a class).