Opening Discussion

- IcP solutions.
- Minute essay comments:
  - Other ways of detecting the button with a pattern.
  - Why would you pick different options?
  - Study sheet for final.
You can use patterns to pull out parts of XML or match on different types of nodes.

Simply put the variable names you want inside of curly braces.

val <a>{s}</a> = node
List and Collection Patterns

- You can also make patterns with collections.
  - case Array(a,b,c) => // use a, b, and c
- Even more cool is what you can do with Lists.
  - case h::t => // h is head and t is tail
  - case a::b::Nil => // two element List
- This can be ideal for recursive methods on lists.
  - def len(lst:List[Int]) = lst match {
    - case Nil => 0
    - case h::t => 1+len(t)
  }
Patterns Everywhere

- Patterns are used in a lot of places in Scala, not just cases and matches.
- The initial declaration of variables is a pattern match. That is why we could assign from tuples.
- The “variable name” in a for loop is actually a pattern. If the pattern isn't matched by an element, that element is skipped.
The Scala collections library is a lot richer than just Lists and Arrays.

I want to introduce three other types of collections to you as they can make your life a lot easier for certain tasks.

They are all parametric so they can work on a variety of types.
Sets

- This is a collection that isn't ordered and doesn't allow duplicates.
- There are both mutable and immutable sets. By default you get the immutable version.
Buffers

- A buffer is a sequence, like an array or a list, but it is mutable like an array and grows like a list.
- You find these in the scala.collection.mutable package.
Maps

- This collection type has two type parameters for a key and a value type.
- You store values and look them up by key.
- The keys are unique.
- There are both mutable and immutable maps. By default, you get the immutable version.
Previously we used recursion to create iteration. This is done with a recursive method that calls itself once and can often be done better with loops.

The real power of recursion comes in when the method calls itself two or more times.

The call stack provides memory so recursion can do one thing, then come back and do another.
Fibonacci Numbers

- The simplest example of a recursive function that calls itself more than once is the Fibonacci numbers.
  - 1, 1, 2, 3, 5, 8, 13, 21, ...
- Each number is the sum of the two before it.
  - $f(n) = \text{if } (n > 2) f(n-1) + f(n-2) \text{ else } 1$
- Simple, but not great.
A classic example of recursion is solving the Towers of Hanoi.

This game is generally made with disks and three pegs.

You need to move the disks from one peg to another.

- Can only move one disk at a time.
- Can't place a disk on one smaller than it.

Solution to N disks: move N-1 disks, move 1 disk, move N-1 disks.
Mazes

- My favorite example is mazes.
- Consider a maze as a 2-D grid with each square either filled or not.
- Now the challenge is to find the length of the shortest path through the maze.
- How do you do that?
What questions do you have about stuff?

Interclass Problem:

- Convert the weather data CSV to XML (with a program).
- Or
- Use all three sorts to sort some type of case class by a field.