Opening Discussion

- Do you have any questions about the quiz?
- Limits of RegEx:
  - Yes, there are limits.
  - Your understanding of them can be a part.
  - Readability can be an issue.
- RegEx are pretty much limited to regular grammars.
There are times when you might want to include elements in your programs that go beyond regular grammars.

An example of this would be an internal DSL (Domain Specific Language). This is like a little language that is understood in your program.

Mathematical formulas count as these, but so would simple commands that have some structure to them.
Here is a CF grammar for math expressions:

- `expr ::= term { “+” term | “-” term }`
- `term ::= factor { “*” factor | “/” factor }`
- `factor ::= floatingPointNumber | (“(“ expr “)”)`

Use `{}` for 0 or more and `[]` for 0 or 1.

Lots of languages here:

- http://www.antlr.org/grammar/list
Scala Parsers

- import scala.util.parsing.combinator._
- class Arith extends JavaTokenParsers {
  - def expr:Parser[Any] = term~rep("+"~term | "-"~term)
  - def term:Parser[Any] = factor~rep("*"~factor | "/"~factor)
  - def factor:Parser[Any] = floatingPointNumber | "("~expr~")"
  
  }
Conversion Rules

- Put in a class that extends one of the Parsers.
  - Productions become methods.
  - Results are Parsers. Next class we'll see how to make it more specific than Any.
  - Consecutive symbols are adjoined with ~.
  - The {...} is replaced with rep(...).
  - The [...] is replaced with opt(...).
Using the Parser

- Call parseAll or parse on your class.
- Takes two arguments:
  - First argument is the parser to use.
  - Second argument is the string to parse.
- Let's code this all up and see it in action.
Questions? Can you think of anyplace you might use this?