# Week 4 Exercises

1. Although mathematicians prefer to work in radians, most scientists and engineers find it easier to think in degrees. Write sine, cosine and tangent methods that accept their arguments in degrees.
2. Write the corresponding set of inverse trigonometric methods that return their values in degrees instead of radians.
3. The math library is missing secant, cosecant and cotangent methods. Write them.
4. The math library lacks a log10 method for taking the common logarithm. Write one.
5. Computer scientists often use a log2 (log base 2). java.lang.Math doesn't have one of those either. Write it.
6. Put all the methods in the previous five exercises into a package and class of your own creation. Be sure to choose sensible, easy-to-understand, hard-to-confuse, names for all packages, classes, and methods. Declare methods and fields static, final, and/or abstract when appropriate.
7. A simple model for the growth of bacteria with an unlimited supply of nutrients says that after t hours an initial population of p0 will have grown to p0 \* e to the 1.4t. Write a Java application that calculates the growth of a colony of bacteria. As usual get the value of p0 and t from the command line.
8. Modify the bacteria growth program so that the time can be input in minutes. Note that the formula still requires a time in hours.
9. Complete the ComplexNumber class discussed in last week's class.
10. Define a reasonably named package for financial classes. Place last week's Money class in this package.
11. Add an overloaded constructor to the Money class that only takes the number of dollars.
12. Add an overloaded constructor to the Money class that takes no arguments and initializes the object to $0.00.
13. Add an equals() method to the Money class.
14. Define an exception class called MoneyOverflowException which can be thrown when an operation with Money results in an over flow. Place this class in the same finance package.
15. Rewrite the methods in the Money class so that they recognize overflow and throw a MoneyOverFlowException if it occurs.
16. Use the classes in the java.math package to eliminate the possibility of overflow in the Money class.
17. Rewrite the two logistic equation problems from Week 2 using the java.math.BigDecimal class to provide 20 decimal digits of precision. Some hints:
    * Invocation of the setScale() method with every iteration is necessary to keep the value of population from overflawing the memory of the computer.
    * You may need to use compareTo() and the Comparable interface instead of either == or equals().

[Start](http://docs.google.com/01.html) | [Previous](http://docs.google.com/69.html) | [Cafe au Lait](http://www.cafeaulait.org/)

Copyright 1997-1999 Elliotte Rusty Harold

[elharo@metalab.unc.edu](mailto:elharo@metalab.unc.edu)

Last Modified February 10, 1999