CSci 555-01 Functional Programming
Assignment #4, Spring 2019

H. Conrad Cunningham

10 April 2019

Assignment #4: Challenge Assignment

Due Friday, 3 May, 11:59 P.M.

General Instructions

All homework and programming exercises must be prepared in accordance with the instructions given in the Syllabus. Each assignment must be submitted to your instructor by its stated deadline.

Citations: In accordance with expected scholarly and academic standards, if you reference outside textbooks, reference books, articles, websites, etc., or discuss an assignment with individuals inside or outside the class, you must document these by including appropriate citations or comments at prominent places in your <submission such as in the header of the primary source file.

Identification: Put your name, course name, and assignment number as comments in each file you submit.

Assignment Description

- This is an individual programming assignment.
- This challenge assignment can be used to replace one of the earlier programming assignments.
- Complete one of the options in the following subsections. If you have done a similar assignment in one of my previous classes, please discuss it with me before undertaking that option.
- Please format and document your program source code appropriately.
- Test your programs appropriately and thoroughly.

- When this assignment is complete, submit your program source code file to Blackboard. Be sure that you identify yourself and the assignment and option in comments in the source file.

Option 1: Sandwich DSL

- This option requires you to complete several exercises from the Sandwich DSL Case Study. You may download the Scala file `SandwichDSL_base.scala` to begin your work.
- Graduate students: Implement all programming exercises in Exercise Sets 1 and 2 except that you may choose to *omit one* of the functions `inOSO`, `intoOSO`, or `eqSandwich`.
- Undergraduate students: Do all the exercises above except you may choose to *omit two* of the functions `inOSO`, `intoOSO`, or `eqSandwich`.

Option 2: Expression-Tree Calculator

- Modify and extend the expression-tree calculator program based on case classes given in the handout Notes on Scala for Java Programmers in the following ways:
  - Change `Const` to represent floating point numbers instead of integers.
  - Add the following new kinds of nodes—`Sub`, `Prod`, and `Div` for subtraction, multiplication, and division of values, respectively; `Neg` for negating a value, and `Sin` and `Cos` for the sine and cosine trigonometric functions, respectively.

    Extend functions `eval` and `derive` to support these additions.
  - (Optional, but recommended, for Undergraduates) Add a new operation `simplify` that takes an expression tree (as extended in the previous tasks), simplifies it by evaluating all subexpressions involving only constants (not evaluating variables), and returns the new expression.
- (Challenge) Redefine the `eval` function to return either an `Option[Double]` or an `Either[String,Double]` to better handle errors such as division by zero or reference to an undefiend variable.
- (Challenge) Extend the simplifications in other ways. For example, you could take advantage of mathematical properties such as identity elements (`x * 1 = x`), zeros (`x * 0 = 0`), associativity (`(x + y) + z = x + (y + z)`, and commutativity (`x + 1 = 1 + x`).

- (Bigger Challenge) Modify and extend the expression-tree calculator program based on traditional object-oriented techniques in the same manner as the previous task.

Option 3: Roll Your Own

- Define your own assignment with the approval of the instructor.