CSCI 301, Lab # 1

Spring 2018

Goal: The purpose of this lab is write some simple code in *Scheme*, and to get familiar with the hardware and software that we will be using all quarter, the lab room, your TA, and the sumbmission procedure for Canvas.

Due: Your program, named `lab01.rkt`, must be submitted to Canvas before midnight, Monday, Oct 1.

Unit tests: At a minimum, your program must pass the unit tests found in the file `lab01-test.rkt`. Place this file in the same folder as your program, and run it; there should be no output.

Program: Write a SCHEME procedure called `make-pi` to compute π using the (slowly converging) series:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots$$

The procedure takes one parameter, which will be the accuracy we need. We can stop whenever the next factor we would add is smaller than this accuracy. For example:

```
(make-pi 0.1) =>  3.09162380666784
(make-pi 0.001) =>  3.1410926536210413
(make-pi 0.0000001) => 3.141592603589817
```

Be careful! This series converges *very* slowly. If you're curious, instrument the procedure so that it also prints out the number of iterations it took to get the accuracy (this is optional).

Make sure that your program *returns* the value of π computed, and doesn't just print it. For example, this should not give an error: `(+ (make-pi 0.1) (make-pi 0.1))`

To get the loop done, define a recursive procedure with (at least) three parameters that behave like this (I've truncated the decimals):

Numerator	Denominator	Sum
4.0	1.0	0.0
-4.0	3.0	4.0
4.0	5.0	2.666
-4.0	7.0	3.466
4.0	9.0	2.895
-4.0	11.0	3.339
4.0	13.0	2.976
-4.0	15.0	3.283
4.0	17.0	3.017
-4.0	19.0	3.252
4.0	21.0	3.041

Make sure your program starts its loop with floating point numbers, *e.g.* 4.0, 1.0, *etc.* If you start with exact integers, Scheme will try to keep exact rational numbers through all of those computations and it will be substantially slower. Also, your answers will look like this:

```
(make-pi 0.1) =>
    516197940314096/166966608033225
```

Make sure to follow the general rules about functional programming: write lots of functions, no assignment statements, no `for` or `while` loops, use `cond` instead of `if`.

Name your program `lab01.rkt` and include a comment block like the one shown (with your own name and W number). Also, hopefully, with the correct code, although this one will pass the unit tests!

lab01.rkt

```
#lang racket
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CSCI 301, Spring 2018
;;
;; Lab #1
;;
;; Your Name Here
;; W012345678
;;
;; The purpose of this program is to
;; bla bla bla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(provide make-pi)

(define make-pi
    (lambda (tolerance) 3.1415))
```