CSCI 301 - Lab # 1

Spring 2025

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, and your TA.

You will submit your program, named `lab1.rkt`, to Canvas.

Unit tests: At a minimum, your program must pass the unit tests found in the file `lab1-test.rkt`. It will report any error; 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
```

This series converges *very* slowly. If you are curious, instrument the procedure so that it also prints the number of iterations it took to get the accuracy.

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 accomplish this, 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 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
```

Some general tips about functional programming: write lots of functions; no assignment statements; no `for` or `while` loops; use `cond` in place of nested chains of `if`s. Remember that you can have DrRacket auto-indent your code for you; please do this to make sure you're following conventional indentation practices.

For commenting, it's a good idea to have a comment block at the top as in the example shown below. For each function you write, give a precise specification for the function in a comment block that starts with two semicolons (`;;`). Inline comments, as needed, can be included within functions, preceded by a single semicolon (`;`).

Name your program `lab1.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!

lab1.rkt

```
#lang racket
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CSCI 301, Spring 2025
;;
;; Lab #1
;;
;; Your Name Here
;; W012345678
;;
;; The purpose of this program is to
;; bla bla bla
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(provide make-pi)

;; return a number within tolerance of pi
(define make-pi
  (lambda (tolerance) pi))
```