CSE1729 - Introduction to Programming	February 18, 2017
Problem Set 4	

1. Consider the harmonic numbers $H_n = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$. Last week you wrote a recursive SCHEME function (named `harmonic`) which, given a number n, computes H_n. Revise your `harmonic` function to take advantage of the `sum` function seen in class and shown below:

```
(define (sum f n)
    (if (= n 0)
        (f 0)
        (+ (f n) (sum f (- n 1)))))
```

   Of course, your new and improved definition should not be recursive and should rely on `sum` to do the hard work.

2. Recall the definition of the derivative of a function from calculus (you will not need to know any calculus to solve this problem):

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

   By choosing very small values for h in the above equation, we can get a good approximation of $f'(x)$.

   (a) Write a function (perhaps call it `der` for *derivative*) in SCHEME that takes a function f and a value for h as formal parameters and returns the function g defined by the rule

$$g(x) = \frac{f(x+h) - f(x)}{h}.$$

   (As mentioned above, for small `h`, g is a good approximation for the derivative of f. Important note: Your function should take a *function* and a number `h` as arguments, and return a *function*.)

   (b) Now take it a step further and write a function which take three formal parameters f, n and h and returns an approximation of the n^{th} derivative of f. Make use of the derivative function you just wrote. (Specifically, you wish to return the function obtained by applying `der` to your function n times.)

   (c) Plot and compare the derivative of $\sin(x)$, as computed by your function with `h` $= .5$, with the function $\cos(x)$. (Use the Racket `plot` package.)

   (d) Now try plotting the *sixteenth* derivative of sin (usually denoted $\sin^{(16)}(x)$). This takes awhile. Why is it so much slower than the plot you computed above?

3. *Newton's Method* is an iterative method for finding successively better approximations to the roots (that is, the zeroes) of a real-valued function. To be more precise, given a function f, Newton's Method is an approach to find a value x for which $f(x) \approx 0$. Newton's Method requires an initial guess for the root (x_0) and determines a sequence of values $x_1, x_2, \ldots$ defined by the recursive rule:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

   For many functions of interest, these "iterates" converge very quickly to a root.

   For example, consider the polynomial $p(x) = x^2 - x - 1$. It turns out that the positive root of p is the number 1.618... that you computed in many ways on the last problem set (the golden ratio). Let's see Newton's method in action. It turns out that the derivative of p is the polynomial $p'(x) = 2x - 1$. (You don't need to know how to determine the explicit derivative of functions for this problem—you'll be using instead the code you generated in the last problem.) Starting with the guess $x_0 = 2$, we may run Newton's method forward to find that:

$$\begin{aligned}
x_0 &= 2\\
x_1 &= x_0 - \frac{p(x_0)}{p'(x_0)} = 2 - \frac{p(2)}{p'(2)} = \frac{5}{3} = 1.666\ldots\\
x_2 &= x_1 - \frac{p(x_1)}{p'(x_1)} = \frac{5}{3} - \frac{p(5/3)}{p'(5/3)} = 1\tfrac{13}{21} = 1.61904\ldots\\
&\ldots
\end{aligned}$$

Write a SCHEME function that implements Newton's Method. Your function should accept three formal parameters, a function f, an initial guess for the root x_0 and the number of iterations to run, n. You can make use of the derivative function from the previous problem (with `h` set to, say, .01).

(a) Demonstrate that your implementation can find the root of the function $f(x) = 2x + 1$.

(b) Demonstrate that your solution can find a good approximation to the golden ratio by working with the polynomial $g(x) = x^2 - x - 1$ (start with the guess 2).

4. You can use your Newton Method solver to extract square roots! Yep, it's true. Note that finding a square root of a fixed number n is the same thing as finding a root of the equation $x^2 - n$. Make a function `sqrt-newt` that takes in one argument `n` and computes an approximation to the square root of `n` by running Newton's method on the polynomial $x^2 - n$ for 40 steps, starting with the guess 1.

5. SICP, Problem 1.29 (Simpson's rule). Recall that the *integral* of a function f between a and b (with $a < b$) is the area underneath the function on the interval $[a, b]$. See Figure 1. Simpson's rule, described in your book, is a method for *approximating* this value.

   To begin with, define a function `(sum term a b)` that takes a function `term` and two integers a and b as arguments and returns the sum `term`(a) + `term`$(a+1) + \cdots +$ `term`(b). Use this in your solution by defining a function that computes the Simpson's rule terms and passing this to `sum`.

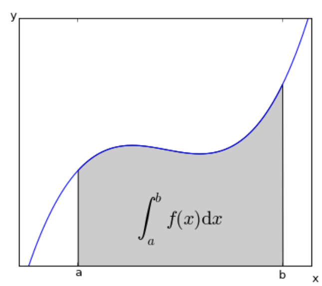

Figure 1: The integral of f from a to b is the area of the shaded region above.

To check your solution, you might try integrating the functions $f_1(x) = x$, $f_2(x) = x^2$, and $f_4(x) = x^4$ on the interval $[0, 1]$ (so $a = 0$ and $b = 1$). You should find that the integral of f_1 is $1/2$ (the area of the triangle), the integral of f_2 is $1/3$, and the integral of f_4 is $1/5$.