Due Thursday October 14, 11:59 p.m. Don't wait until the last second!

Read each question carefully and write a short procedure to implement the function described.

Check that the language level in Dr.Scheme is set to R5RS. Then put all the procedures that you write into one Scheme file named hw2.scm. **Be sure to name the procedures as requested in the question**. This way it is easy to run standard test cases. A limited set of test cases is available in hw2-test-it.scm, provided along with this writeup. A sample execution of hw2-test-it is shown on the last page of this handout.

There are three steps to turning in your homework:
a.) Use the turnin link from the calendar page; follow the instructions and submit the file hw2.scm.
b.) Then complete the web survey that you are directed to.
c.) **Print a copy of your hw2.scm and turn it in during class on Friday October 15. Write your name and UW Net ID prominently on the first page.**

1. The Fibonacci numbers are defined as follows:
   fib(0) = 0
   fib(1) = 1
   fib(n) = fib(n-1) = fib(n-2)

   a.) Write a function (fib-recursive n) that calculates fib(n) while generating a *recursive process.*.
   b.) Write a function (fib-iterative n) that calculates fib(n) while generating an *iterative process.*
   Recall that your process will be iterative if you use tail recursion.

2. In lecture we defined the following abstract function:

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a) (sum term (next a) next b))))
```

   Define a new function
   `(define (filter-sum term a next b filter)`
   that serves the same purpose but only adds terms to the sum if `filter` returns true for the specified index value. For instance, you could use this function to add up all the odd numbers from 1 to 20 with:

```
(define (identity x) x)
(define (inc i) (+ i 1))
(define (odd i) (= 1 (remainder i 2)))
(filter-sum identity 1 inc 20 odd)
```

(see the rest of this question on the next page)

a.) Write `filter-sum` as defined above.
b.) Write a new function `(sum-fibs n)` that uses `filter-sum` to sum all the Fibonacci numbers that are less than n (**not** the first n Fibonacci numbers). Your answer is not required to be very efficient, but should complete quickly on the sample inputs.
c.) Write a new function `(sum-rel-primes n)` that uses `filter-sum` to sum all the positive integers less than n that are relatively prime to n (i.e., all positive integers i < n such that GCD(i,n) = 1). Use your `euclid` function from Homework #1.

3. (adapted from Abelson, Sussman, and Sussman)
Using Simpson's Rule, the integral of a function f between a and b is approximated as:

$$\frac{h}{3}[y_0 + 4y_1 + 2y_2 + 4y_3 + .2y_4 + ... + 4y_{n-1} + y_n]$$

(the 2's and 4's keep alternating until the end) where h = (b-a) / n, for some even integer n, and y_k = f(a + kh). (Increasing n increases the accuracy of the approximation.) Define a procedure `(simpson f a b n)` to perform this calculation. For instance, the integral of the `cube` function between 0 and 1 is ¼, though your calculation will not produce this exact answer. You should use `let` at least once in your answer.

4. Write a function `(apply-f-g-ntimes f g n)` that returns a function that takes one number argument x and alternately applies f and g to x so that in the end f and g have each been applied n times. The function f is applied first, so (apply-f-g-ntimes f g 2) should produce a function that computes g(f(g(f(x)))).

5. (adapted from Abelson, Sussman, and Sussman)
The idea of *smoothing* a function is an important concept in signal processing. If *f* is a function and *dx* is some small number, then the smoothed version of *f* is the function whose value at a point *x* is the average of *f*(*x* - *dx*), *f*(*x*), and *f*(*x* + *dx*). Write a function `(smooth f dx)` that takes as input a procedure that computes *f* and a number dx, and returns a function that computes the smoothed *f*.

```
fib-recursive
1 => 1 : 0
13 => 13 : 0
55 => 55 : 0

fib-iterative
1 => 1 : 0
13 => 13 : 0
55 => 55 : 0
354224848179261915075 => 354224848179261915075 : 0

filter-sum
100 => 100 : 0
210 => 210 : 0

sum-fibs
20 => 20 : 0
2583 => 2583 : 0

sum-rel-primes
20 => 20 : 0
126 => 126 : 0
2000 => 2000 : 0

simpson
0.25 => 0.24999999999999997 : 2.7755575615628914e-017
0 => -4.4101190598976977e-016 : 4.4101190598976977e-016
12.5 => 12.5 : 0.0

apply-f-g-ntimes
25 => 25 : 0
458329 => 458329 : 0
7 => 7 : 0

smooth
1 => 1.0 : 0.0
-1.005 => -1.005 : 0.0
16.001666666666665 => 16.001666666666665 : 0.0
```