Whenever you write any procedure for this class, the procedure should be tested thoroughly. For problems from Chapter 1, include those tests. For problems from other chapters, I will allow a redo when I find a bug only if you have shown your testing. In other words, if you have complete confidence that your testing was sufficiently thorough, you need not include your tests.

When a solution to a problem is a program, provide an explanation of the program if the logic is at all confusing. Generally, your programs should be easy to read without any explanation due to clear formatting and good choice of procedure and parameter names. When a solution is anything else, you should never simply provide an answer (such as a number); some explanation is required to justify your answer.

1a: (a) Write a Scheme expression with no multidigit numbers which has 3.14 as its value. (For the purposes of this problem, the number .02 has two digits.)
(b) Write a Scheme expression with no multidigit numbers which has 22/7 as its value.

1b: [HKK 1.4] According to the Joy of Cooking, candy syrups should be cooked 1 degree cooler than listed in the recipe for each 500 feet of elevation above sea level.

(a) Define candy-temperature to be a procedure that takes two arguments: the recipe’s temperature in degrees and the elevation in feet. It should calculate the temperature to use at that elevation. The recipe for Chocolate Caramels calls for a temperature of 244 degrees; suppose you wanted to make them in Denver, the “mile high city.” (One mile equals 5280 feet.) Use your procedure to find the temperature for making the syrup.
(b) Candy thermometers are usually calibrated only in integer degrees, so it would be handy if the candy-temperature procedure would give an answer rounded to the nearest degree. Rounding can be done using the predefined procedure called round. For example, (round 7/3) and (round 5/3) both evaluate to 2. Insert an application of round at the appropriate place in your procedure definition and test it again.

1c: [HKK 1.7] Write a succinct English description of the effect of each of the following procedures. Try to express what each calculates, not how it calculates that.

(a) (define puzzle1
  (lambda (a b c)
    (+ a (if (> b c)
      b
      c))))
(b) (define puzzle2
  (lambda (x)
    ((if (< x 0)
      -
      +)
     0 x)))

1d: [HKK 1.16] A 10-foot-long ladder leans against a wall, with its base 6 feet away from the bottom of the wall. How high on the wall does it reach? This question can be answered by evaluating (ladder-height 10 6) after entering the following definition. Make a diagram such as the one in Figure 1.1 showing the evaluation of (ladder-height 10 6) in the context of this definition:

(define ladder-height
  (lambda (ladder-length base-distance)
    (sqrt (- (square ladder-length)
      (square base-distance))))))

1e: If n points are placed on a circle, there are \( \binom{n}{2} \) ways of choosing a pair of points to connect, where

\[
\binom{n}{2} = \frac{n(n-1)}{2}.
\]
For example, if $n = 4$, $\binom{4}{2} = \frac{4(4-1)}{2} = 6$. The 6 lines are shown below:

![Diagram of 4 lines forming a square]

The number of possible pairs of lines would be $\binom{\binom{n}{2}}{2}$. For $n = 4$,

$$\binom{\binom{4}{2}}{2} = \binom{6}{2} = 15.$$

The possible 15 pairs of lines are:

```
X    Y    Z    W
\|   \|   \|   \|
/     /     /     /
\     \     \     \
/    \    /    \    /
```

(a) Write a procedure `n-choose-2` that takes one argument, $n$, and computes $\binom{n}{2}$. Test your procedure on several inputs including 3, 4 and 5.

(b) Write a procedure `n-choose-2-choose-2` to compute $\binom{\binom{n}{2}}{2}$ using the procedure `choose`. Test your procedure on several inputs including 3, 4 and 5.

2a: Recall from problem 1e that $\binom{n}{2}$ is the number of pairs possible among $n$ points. Phrased differently, if $n$ people are at a party, and everyone shakes hands with everyone else, then $\binom{n}{2}$ handshakes will occur at the party.

Suppose, now, that you did not know the formula for $\binom{n}{2}$. Let’s count the handshakes a different way. Each time a person arrives at the party, the person shakes hands with everyone else who was already there. Use this description to write a recursive program to compute $\binom{n}{2}$.

2b: The following definition makes use of `quotient`, which is a procedure built in to Scheme. The `quotient` procedure divides one integer by another, but returns only the quotient, ignoring the remainder. For example, `(quotient 2718 10)` evaluates to 271, because 271 is the number of times 10 goes into 2718. Your task is to list the `(round-down 2718)`, using the same format as at the bottom of page 26 in your textbook.

```
(define round-down
  (lambda (n)
    (if (< n 10)
        n
        (* (round-down (quotient n 10))
          10)))))
```

2c: [HKK 2.10] Write a procedure that calculates the number of odd digits in an integer. (Reminder: There is a built-in predicate called `odd?`.)

2d: [HKK 2.16] Consider the following procedure `foo`:

```
(define foo
  (lambda (x n)
    (if (= n 0)
        1
        (+ (expt x n) (foo x (- n 1))))))
```
Use induction to prove that \((\text{foo } x \ n)\) terminates with the value 
\[
\frac{x^{n+1} - 1}{x - 1}
\]
for all values of \(x \neq 1\) and for all integers \(n \geq 0\). You may assume that \texttt{expt} works correctly, (i.e., \((\texttt{expt } b \ m)\) returns \(b^m\)). \textit{Hint:} The inductive step will involve some algebra.

2e: [HKK 2.19] Prove that for all nonnegative integers \(n\) the following procedure computes the value \(2^{(2^n)}\):

\[
\begin{align*}
\text{(define foo} \\
\text{  (lambda (n)} \\
\text{  (if (= n 0)} \\
\text{    2} \\
\text{      (expt (foo (- n 1)) 2)))})
\end{align*}
\]

\textit{Hint:} You will need to use certain laws of exponents, in particular that \((2^a)^b = 2^{ab}\) and \(2^a2^b = 2^{a+b}\).

3a: (HKK 2.12 and 3.2) Your goal is to write two procedures, one iterative and one recursive, for finding the number of \(2\)'s in the prime factorization of a positive integer. In other words, if the integer \(n\) can be written \(2^i \times k\) for \(k\) odd, your procedure would return \(i\).

(a) Write a procedure which generates a recursive process.

(b) Write a procedure which generates an iterative process.

(c) Show how your two procedures work when given the argument 24 using the substitution model. Explain why your first procedure is iterative and the second is recursive.

3b: (HKK 3.13) Does the following procedure generate a recursive process or an iterative process? Clearly justify your answer.

\[
\begin{align*}
\text{(define largest-odd-divisor} \\
\text{  (lambda (n)} \\
\text{    (if (odd? n)} \\
\text{      n} \\
\text{       (largest-odd-divisor (/ n 2))))})
\end{align*}
\]

3c: [HKK 3.17] \textit{Falling factorial powers} are similar to normal powers and also similar to factorials. We write them as \(n^\underline{k}\) and say “\(n\) to the \(k\) falling.” This means that \(k\) consecutive numbers should be multiplied together, starting with \(n\) and working downward. For example, \(7^\underline{3} = 7 \times 6 \times 5\) (i.e., three consecutive numbers from 7 downward multiplied together).

Write a procedure for calculating falling factorial powers that generates an iterative process.

4a: [HKK 4.11] Consider the following procedures:

\[
\begin{align*}
\text{(define factorial} \\
\text{  (lambda (n)} \\
\text{    (if (= n 0)} \\
\text{      1} \\
\text{       (* n (factorial (- n 1))))})
\end{align*}
\]

\[
\begin{align*}
\text{(define factorial-sum1 ; returns 1! + 2! + \ldots + n!} \\
\text{  (lambda (n)} \\
\text{    (if (= n 0)} \\
\text{      0} \\
\text{      (+ (factorial n))})
\end{align*}
\]
(factorial-sum1 (- n 1)))

(define factorial-sum2 ; also returns 1! + 2! + ... + n!
  (lambda (n)
    (define loop
      (lambda (k fact-k addend)
        (if (> k n)
          addend
          (loop (+ k 1)
            (* fact-k (+ k 1))
            (+ addend fact-k)))))))

In answering the following, assume that \( n \) is a nonnegative integer. Also, justify your answers.

(a) Give a formula for how many multiplications the procedure `factorial` does as a function of its argument \( n \).
(b) Give a formula for how many multiplications the procedure `factorial-sum1` does (implicitly through `factorial`) as a function of its argument \( n \).
(c) Give a formula for how many multiplications the procedure `factorial-sum2` does as a function of its argument \( n \).

4b: Where possible, fill in the blanks in the following statements with positive real numbers so as to make the statement true. Where impossible, put a question mark in the blank that you cannot truthfully fill in and explain why it cannot be filled in.

(a) (Sample) \( n - 1 \) is \( \Theta(n) \) because for any \( n \geq 2 \), we know that \( \frac{1}{2} \cdot n \leq n - 1 \leq \frac{3}{2} \cdot n \).
(b) \( 3n \) is \( \Theta(n) \) because for any \( n \geq \ldots \), we know that \( \ldots \cdot n \leq 3n \leq \ldots \cdot n \).
(c) \( 3n - 20 \) is \( \Theta(n) \) because for any \( n \geq \ldots \), we know that \( \ldots \cdot n \leq 3n \leq \ldots \cdot n \).
(d) \( n^3 + 6n^2 \) is \( \Theta(n^3) \) because for any \( n \geq \ldots \), we know that \( \ldots \cdot n^3 \leq n^3 + 6n^2 \leq \ldots \cdot n^3 \).
(e) \( n^3 + 6n^2 \) is \( \Theta(n) \) because for any \( n \geq \ldots \), we know that \( \ldots \cdot n \leq n^3 + 6n^2 \leq \ldots \cdot n \).

4c: [HKK 4.13] Consider the following procedure:

```
(define bar
  (lambda (n)
    (cond ((= n 0) 5)
          ((= n 1) 7)
          (else (* n (bar (- n 2)))))))
```

How many multiplications (expressed in \( \Theta \) notation) will the computation of `(bar n)` do? Justify your answer. You may assume that \( n \) is a nonnegative integer.

4d: [HKK 4.14] Consider the following procedures:

```
(define foo
  (lambda (n) ; computes n! + (n!)^n
    (+ (factorial n) ; that is, (n! plus n! to the nth power)
      (bar n n)))))

(define bar
  (lambda (i j) ; computes (i!)^j (i! to the jth power)
    (if (= j 0)
      1
      (* (factorial i)))
```
(bar i (- j 1))))

(define factorial
  (lambda (n)
    (if (= n 0)
      1
      (* n (factorial (- n 1))))))

How many multiplications (expressed in $\Theta$ notation) will the computation of $(\text{foo } n)$ do? Justify your answer. Be sure to count multiplications done by either of the other procedures.

5a: [HKK 5.15] Write a higher-order procedure called make-function-with-exception that takes two numbers and a procedure as parameters and returns a procedure that has the same behavior as the procedural argument except when given a special argument. The two numerical arguments to make-function-with-exception specify what that exceptional argument is and what the procedure made by make-function-with-exception should return in that case. For example, the usually-sqrt procedure that follows behaves like $\sqrt{\cdot}$, except that when given the argument 7, it returns the result 2:

(define usually-sqrt
  (make-function-with-exception 7 2 sqrt))

(usually-sqrt 9)
3

(usually-sqrt 16)
4

(usually-sqrt 7)
2

5b: [HKK 5.16] If two procedures $f$ and $g$ are both procedures of a single argument such that the values produced by $g$ are legal arguments to $f$, the composition of $f$ and $g$ is defined to be the procedure that first applies $g$ to its argument and then applies $f$ to the result. Write a procedure called compose that takes two one-argument procedures and returns the procedure that is their composition. For example, $((\text{compose } \sqrt{} \text{ abs}) -4)$ should compute the square root of the absolute value of $-4$.

5c: [HKK 5.20] Suppose the following have been defined:

(define f
  (lambda (m b)
    (lambda (x) (+ (* m x) b))))

(define g (f 3 2))

For each of the following expressions, indicate whether an error would be signaled, the value would be a procedure, or the value would be a number. If an error is signaled, explain briefly the nature of the error. If the value is a procedure, specify how many arguments the procedure expects. If the value is a number, specify which number.

(a) f
(b) g
(c) (* (f 3 2) 7)
(d) (g 6)
(e) (f 6)
5d: Write a procedure \( (\text{estimate-integral } f \ a \ b) \) which estimates the integral \( \int_{a}^{b} f(x) \, dx \). (Do not panic if you have not had calculus.) One way to do this is by a Riemann sum. For example,

\[
\int_{2}^{5} f(x) \, dx \approx .01 \cdot [f(2.00) + f(2.01) + f(2.02) + \cdots + f(4.99)]
\]

More generally,

\[
\int_{a}^{b} f(x) \, dx \approx \delta \cdot [f(a) + f(a + \delta) + f(a + 2\delta) + f(a + 3\delta) + \cdots + f(a + n\delta)]
\]

where \( \delta \) is a small positive number and \( a + n\delta \approx b \).

(a) Write the procedure \( (\text{estimate-integral } f \ a \ b) \), fixing \( \delta = .01 \) as in the first example above. Refer to any calculus textbook to identify a few test cases.

(b) Write the procedure \( (\text{estimate-integral-with-step } f \ a \ b \ \delta) \) which takes the step \( \delta \) as an argument.

(c) Write a procedure \( (\text{make-integrater } \ \delta) \) which takes an argument \( \delta \) and returns a procedure. That procedure should accept three arguments \( f, a, \) and \( b \), and estimate \( \int_{a}^{b} f(x) \, dx \) using a Riemann sum with step \( \delta \).

(d) Optional: Can you find functions for which this Riemann sum gives a very bad estimate? Can you improve the program to work with some of these troublesome functions?

6a: [HKK 6.3] The version of Nim we have just written designates the winner as the one taking the last coin. What needs to be changed in order to reverse this, that is, to designate the one taking the last coin as the loser?

7a: [HKK 7.44 with the last part changed] Consider the following two procedures. The procedure \textit{last} selects the last element from a list, which must be nonempty. It uses \textit{length} to find the length of the list.

\begin{verbatim}
(define last
  (lambda (lst)
    (if (= (length lst) 1)
      (car lst)
      (last (cdr lst))))

(define length
  (lambda (lst)
    (if (null? lst)
      0
      (+ 1 (length (cdr lst))))))
\end{verbatim}

As always, an explanation is appropriate for each of the following questions; when counting \texttt{cdrs} be sure to count those done by either procedure.

(a) How many \texttt{cdrs} does \((\text{length } lst)\) do when \( lst \) has \( n \) elements?

(b) How many calls to \text{length} does \((\text{last } lst)\) make when \( lst \) has \( n \) elements?

(c) Express in \( \Theta \) notation the total number of \text{cdrs} done by \((\text{last } lst)\), including \text{cdrs} done by \text{length}, again assuming that \( lst \) has \( n \) elements.

(d) Write a new version of \textit{last} that is more efficient, in the sense that the number of \text{cdrs} is improved by more than merely a constant factor, as expressed by \( \Theta \)-notation. Express the number of \text{cdrs} performed by your improved version using \( \Theta \)-notation.
7b: [HKK 7.46] Write a higher-order procedure \textbf{make-list-scaler} that takes a single number \textit{scale} and returns a procedure that, when applied to a list \textit{lst} of numbers, will return the list obtained by multiplying each element of \textit{lst} by \textit{scale}. Thus, you might have the following interaction:

\[
\text{(define scale-by-5 (make-list-scaler 5))}
\]
\[
\text{(scale-by-5 '(1 2 3 4))}
\]
\[
(5 10 15 20)
\]

7c: [HKK 7.50] Consider the following procedure (together with two sample calls):

\[
\text{(define repeat}
\]
\[
\text{(lambda (num times)}
\]
\[
\text{\hspace{1cm} (if (\text{=} \text{times} 0)}
\]
\[
\text{\hspace{1cm} '())}
\]
\[
\text{\hspace{1cm} (cons num (repeat num (- times 1)))))}
\]
\[
\text{(repeat 3 2)}
\]
\[
(3 3)
\]
\[
\text{(repeat 17 5)}
\]
\[
(17 17 17 17 17)
\]

(a) Explain why \textit{repeat} generates a recursive process.

(b) Write an iterative version of \textit{repeat}.

7d: [HKK 7.49] Given a predicate that tests a single item, such as \texttt{positive?}, we can construct an “all are” version of it for testing a list; an example is a predicate that tests whether all elements of a list are positive. Define a procedure \texttt{all-are} that does this; that is, it should be possible to use it in ways like the following:

\[
\text{((all-are positive?) '(1 2 3 4))}
\]
\[
\text{\#t}
\]
\[
\text{((all-are even?) '(2 4 5 6 8))}
\]
\[
\text{\#f}
\]

7e: [HKK 7.19] When our children started bringing home dozens of cheap plastic spiders, we asked them to restrict themselves to getting only one of each kind of prize. Write a procedure that is given the prize list and amount and computes the number of prize combinations that you can buy using exactly that amount and assuming that you can’t get more than one of any particular prize.

7f: [HKK 7.20] Write another procedure that will determine the number of combinations you can buy using no more than a maximum amount of tickets while still insisting that you can have at most one of each kind of prize. As in 7.19, assume that only one of each prize is allowed.

For the next few problems, when testing procedures on trees, you may choose to “cheat” and type in the tree at the top of page 251 directly (circumventing the ADTs \texttt{make-nonempty-tree} and \texttt{make-empty-tree}):

\[
\text{(define tree '}(4 (2 (1 () ()) (3 () ())) (6 (5 () ()) (7 () ()))))}
\]

8a: [HKK 8.4 expanded] Eliminate \texttt{append} by using an “onto” parameter to make a more efficient version of \texttt{inorder}. Explain why the new version is more efficient.
8b: [HKK 8.30] Fill in the following definition of the procedure `successor-of-in-or`. This procedure should take three arguments: a value (`value`), a binary search tree (`bst`), and a value to return if no element of the tree is larger than `value` (`if-none`). If there is any element, `x`, of `bst` such that `x > value`, the smallest such element should be returned. Otherwise, `if-none` should be returned.
(define successor-of-in-or
  (lambda (value bst if-none)
    (cond ((empty-tree? bst)
             ((<= (root bst) value)
              (successor-of-in-or
               value))
           (else
            (successor-of-in-or
             (successor-of-in-or
              value))))))))

8c: [HKK 8.31] Write a procedure that takes as arguments a binary search tree of numbers, a lower bound, and an upper bound and counts how many elements of the tree are greater than or equal to the lower bound and less than or equal to the upper bound. Assume that the tree may contain duplicate elements. Make sure your procedure doesn’t examine more of the tree than is necessary.

9a: [HKK 9.5] Rewrite list->sequence so that it has a branch for sequence reference.

9b: [HKK 9.20] Global Amalgamations Corp. has just acquired yet another smaller company and is busily integrating the data processing operations of the acquired company with that of the parent corporation. Luckily, both companies are using Scheme, and both have set up their operations to tolerate multiple representations. Unfortunately, one company uses operation tables as type tags, and the other uses procedural representations (i.e., message passing). Thus, not only are multiple representations now co-existing, but some of them are type-tagged data and others are message-passing procedures. You have been called in as a consultant to untangle this situation.

What is the minimum that needs to be done to make the two kinds of representation happily coexist? Illustrate your suggestion concretely using Scheme as appropriate. You may want to know that there is a built-in predicate pair? that tests whether its argument is a pair, and a similar one, procedure?, that tests whether its argument is a procedure.

9c: [Variant on HKK 9.22] Assume that infinity has been defined as a special number that is greater than all normal (finite) numbers and that when added to any finite number or to itself, it yields itself. (In some Scheme systems you can define it as follows: (define infinity (/ 1.0 0.0)).) Now there is no reason why sequences need to be of finite length.

(a) Write a constructor for some interesting kind of sequence which is infinite in length.
(b) Demonstrate that your infinite sequence can be used together with sequence-cons or Exercise 9.6’s sequence-map to produce a new infinite sequence.