Assignment: 5
Due: Tuesday, February 25, 9:00 pm
Language level: Beginning Student with list abbreviations
Files to submit: debug-a05.rkt, listfun.rkt, movies.rkt, matrixfun.rkt
Warmup exercises: HtDP 13.0.3, 13.0.4, 13.0.7, 13.0.8, 17.2.1, 17.3.1, 17.6.4, 17.8.3
Practice exercises: HtDP 12.4.1, 12.4.2, 13.0.5, 13.0.6, 14.2.3, 14.2.4, 14.2.6, 17.1.2, 17.6.2, 17.8.4

- Make sure you read the OFFICIAL A05 post on Piazza for the answers to frequently asked questions.
- Unless stated otherwise, all policies from Assignment 04 carry forward.
- This assignment covers material up to the end of Module 08.
- Of the built-in list functions, you may use only cons, first, second, third, fourth, fifth, rest, empty?, cons?, list, member?, and length. You may also use equal? to compare two lists.
- If you choose to define a type, provide a complete data definition. You do not need to provide a template, but can have one if you want.
- Remember that basic tests are meant as sanity checks only; by design, passing them should not be taken as any indication that your code is correct, only that it has the right form.
- Unless the question specifically says otherwise, you are always permitted to write helper functions. You may use any constants or functions from any part of a question in any other part.

1. [10% Correctness] Question 1
Consider the code in debug-a05.rkt, which may contain syntax, mathematical, and style errors. Correct the code and style.

2. [20% Correctness] Question 2
Implement the following functions that perform recursion on multiple parameters:

(a) dot-product consumes two lists of numbers and produces their dot product. Note that the two lists must have the same length. The dot product is computed as \(a_0 \cdot b_0 + a_1 \cdot b_1 + \ldots + a_n \cdot b_n\) where \(a_i\) and \(b_i\) are the elements of lists \(a\) and \(b\) respectively. For example, \((\text{dot-product} \ (\text{cons} \ 1 \ (\text{cons} \ 3 \ \text{empty})) \ (\text{cons} \ 2 \ (\text{cons} \ 3 \ \text{empty})))\) should produce 11.
(b) \textit{combine-sorted} consumes two sorted lists of numbers in non-decreasing order and produces a new list that is the combination of the two input lists. The elements of the new list should be in non-decreasing order. Lists may contain duplicates—if an item appears $n$ times in the first list, and $m$ times in the second, then the produced list will contain $n + m$ copies of that item.

For example, \texttt{(combine-sorted (cons 1 (cons 3 empty)) (cons 2 empty))} should produce \texttt{(cons 1 (cons 2 (cons 3 empty)))}.

(c) \textit{my-remove} consumes a list of numbers and a number $n$ and removes all instances of $n$ from that list. For example, \texttt{(my-remove (cons 1 (cons 2 (cons 3 empty))) 2)} should produce \texttt{(cons 1 (cons 3 empty))}.

(d) \textit{list-xor} consumes two lists with elements of any type, $a$ and $b$, and produces a list that contains only the items that are in $a$ or $b$, but no elements that are contained in both $a$ and $b$. For example, \texttt{(list-xor (cons 1 (cons 2 (cons 3 empty))) (cons 1 (cons 4 empty)))} should produce \texttt{(cons 2 (cons 3 (cons 4 empty)))}.

Note: you may assume there are no duplicates in either list for this question and the elements must appear in the produced list in the same order as their source lists, with the items from first list preceding elements of the second.

Place your solutions in \texttt{listfun.rkt}.

3. [30\% Correctness] Question 3

This question deals with movies. Movies having the following data definition:

\begin{verbatim}
;; A Movie is a (list Str Str Nat Num Num).
Or, more simply put, A Movie is a list containing:
\end{verbatim}

- the name of the film
- the directors name
- the year it was released, $1838 \leq \text{year} \leq 2020$
- the production budget, $\geq 0$
- the box office sales, $\geq 0$

(a) Write the template functions for a \texttt{Movie} and a \texttt{(listof Movie)}. Call your templates \texttt{movie-template} and \texttt{listof-movie-template}

(b) Write the function \texttt{sort-by-name} that consumes a \texttt{(listof Movie)} and produces a list of Movies sorted lexicographically by the names of films.

\texttt{Hint:} write a function that consumes a sorted list of Movies and inserts a Movie into the correct place first.

\texttt{Hint:} consider the built-in function \texttt{string<?}, which produces true if the first argument would fall lexicographically before the second.
(c) Write the function `get-by-year` that consumes a `(listof Movie)` and a year and produces a list of movies that were released in that year. The contents of the list should be in the same order as they appear in the consumed list of Movies.

(d) Write the function `find-biggest-flop` that consumes a `(listof Movie)` and produces a `Movie` that is the biggest flop, or the symbol `’none`. A biggest flop is the movie with the greatest difference between production budget and box office sales.

Hint: first write a function that determines if a film is a flop. Then write a function that finds all the flops.

Put your solutions into the file `movies.rkt`.

4. [20% Correctness] Question 4

A matrix is a two-dimensional grid of numbers represented as a list of equal-lengthed, non-empty lists.

```racket
;; A Matrix is one of:
;; * (cons (listof Num) empty)
;; * (cons (listof Num) Matrix)
;; where each (listof Num) has the same length
```

You can add and subtract matrices of the same size by applying the operators element-wise. We can represent a matrix as a list of lists as such: `(list (list 1 2) (list 3 4))`. If we add this matrix to itself it yields: `(list (list 2 4) (list 6 8))`.

Implement the following functions for matrices:

(a) `add-matrices`, which consumes two matrices of equal size and produces a matrix that is the sum of the parameters.

(b) `sub-matrices`, which consumes two matrices of equal size and produces a matrix that is the first matrix minus the second.

(c) `scalar-mult-matrix`, which consumes a matrix and a number and produces a matrix where each element was multiplied by the number.

Place your solutions in `matrixfun.rkt`.

---

**Bonus Question [10%]**

Implement `matrix-multiply` that consumes to equal sized matrices `A` and `B` and produces the matrix `A * B`. For more information about matrix multiplication, read the Wikipedia article at here.

Place your solution into `bonus-a05.rkt`.

---

**Enhancements:** Reminder—enhancements are for your interest and are not to be handed in.
There is a strong connection between recursion and induction. Mathematical induction is the proof technique often used to prove the correctness of programs that use recursion; the structure of the induction parallels the structure of the function. As an example, consider the following function, which computes the sum of the first \( n \) natural numbers.

\[
\text{(define \( \text{sum-first} \ n \))}
\text{(cond}
\text{ \[(\text{zero?} \ n) \ 0\]}
\text{ \[else \ (+ \ n \ (\text{sum-first} \ (\text{sub1} \ n)))\]})
\]

To prove this program correct, we need to show that, for all natural numbers \( n \), the result of evaluating \( \text{sum-first} \ n \) is \( \sum_{i=0}^{n} i \). We prove this by induction on \( n \).

**Base case:** \( n = 0 \). When \( n = 0 \), we can use the semantics of Racket to evaluate \( \text{sum-first} \ 0 \) as follows:

\[
\text{(sum-first} \ 0) \ ; \Rightarrow 
\text{(cond} \ [(\text{zero?} \ 0) \ 0]\text{[else ...]} \) ; \Rightarrow 
\text{(cond} \ [true \ 0]\text{[else ...]} ; \Rightarrow 
0
\]

Since \( 0 = \sum_{i=0}^{0} i \), we have proved the base case.

**Inductive step:** Given \( n > 0 \), we assume that the program is correct for the input \( n - 1 \), that is, \( \text{sum-first} \ (\text{sub1} \ n) \) evaluates to \( \sum_{i=0}^{n-1} i \). The evaluation of \( \text{sum-first} \ n \) proceeds as follows:

\[
\text{(sum-first} \ n) \ ; \Rightarrow 
\text{(cond} \ [(\text{zero?} \ n) \ 0]\text{[else ...]} \) ;(we know \( n > 0 \)) \Rightarrow 
\text{(cond} \ [false \ 0]\text{[else ...]} ; \Rightarrow 
\text{(cond} \ [else \ (+ \ n \ (\text{sum-first} \ (\text{sub1} \ n))) \]) \ ; \Rightarrow 
(+ \ n \ (\text{sum-first} \ (\text{sub1} \ n)))
\]

Now we use the inductive hypothesis to assert that \( \text{sum-first} \ (\text{sub1} \ n) \) evaluates to \( s = \sum_{i=0}^{n-1} i \). Then \( (+ \ n \ s) \) evaluates to \( n + \sum_{i=0}^{n-1} i \), or \( \sum_{i=0}^{n} i \), as required. This completes the proof by induction.

Use a similar proof to show that, for all natural numbers \( n \), \( \text{sum-first} \ n \) evaluates to \( (n^2 + n)/2 \).

**Note:** Summing the first \( n \) natural numbers in imperative languages such as C++ or Java would be done using a \texttt{for} or \texttt{while} loop. But proving such a loop correct, even such a simple loop, is considerably more complicated, because typically some variable is accumulating the sum, and its value keeps changing. Thus the induction needs to be done over time, or number of statements executed, or number of iterations of the loop, and it is messier because the semantic model in these
languages is so far-removed from the language itself. Special temporal logics have been developed to deal with the problem of proving larger imperative programs correct.

The general problem of being confident, whether through a mathematical proof or some other formal process, that the specification of a program matches its implementation is of great importance in safety-critical software, where the consequences of a mismatch might be quite severe (for instance, when it occurs with software to control an airplane, or a nuclear power plant).