General Instructions:

- Several problems required using particular other functions. In many cases, the auto-grader already checked for this, so you should not re-penalize for the same thing. In other cases, it was not feasible to check this automatically. The per-problem instructions will make clear what to look for and what has already been auto-graded.
- Mutation, such as **`set!`** or **`set-mcar!`** is generally poor style, except in problem 10, where it is essential to the solution. In other problems, give at most a 3 to solutions using mutation.

Problem 1:

Here is a sample solution:

```
(define (sequence low high stride)
  (if (> low high)
      null
      (cons low (sequence (+ low stride) high stride))))
```

On this and all problems, do not penalize using the longer form **`(define sequence (lambda (low high stride) ...`** instead of using the syntactic sugar for function definitions as in the sample above.

There is little need for a solution more complicated than the one above, but it is okay to give a 5 to a solution that uses a local helper function to avoid passing **`high`** and **`stride`** recursively. It is also okay to use **`cond`** instead of **`if`** although **`if`** is usually better style when there are only two cases.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 2

Here is a sample solution:

```
(define (string-append-map xs suffix)
  (map (lambda (s) (string-append s suffix)) xs))
```

The auto-grader already penalized solutions that did not use **`string-append`** and **`map`** as helper functions, so we do not need peer assessment to judge this same issue.

There is little benefit to a longer solution here, so probably give at most a 4 to solutions that use some form of **`let`**-expression.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 3

Here is a sample solution:

```
(define (list-nth-mod xs n)
  (cond [(< n 0) (error "list-nth-mod: n must be non-negative")]
        [(null? xs) (error "list-nth-mod: list must be non-empty")]
        [#t (let* ([len (length xs)]
                   [posn (remainder n len)])
              (car (list-tail xs posn)))]))
```