Question 2

remove-outliers

--- Students should mention that the order is preserved in their purpose description.
--- Some students forgot that the consumed list is non-empty.
--- Since it is not possible for every number in the list of numbers to be an outlier, the produced list would be non-empty too. That said, students who used listof or ne-listof still got the mark.
--- Many students did not define local constants to prevent repeated list traversals.
--- Part of the problem was to understand how to translate the presented formulas into Racket code. Although using alternate formulas was allowed, the majority of the students who failed test cases did not directly translate the given formula.

zero-fill

--- The contract needed a requirement that the length of the string was at most 20.
--- You can reconstruct the string with foldr with base case a list of #\0s of the appropriate length.

remove-duplicates

--- The purpose needed to mention order preservation.
--- Many students struggled to get a good intuition for this one. You can use foldr and filter on the result of recursion on the rest of the list!

Question 3

--- Students should use the given data definitions to simplify their contract. Although writing out an equivalent data type is not wrong, students should be used to using the definitions given as they are more specific, simple, and intuitive to the reader.

transpose

--- Missing check for an empty image.
--- There are different ways to do this question, so see the posted solution.

Question 4

--- Many students did not know how to represent functions in contracts. To show that a function consumes/produces another function, write that function’s contract inside round brackets. In addition, make sure you understand the difference between the type X and the type Any and when to use each type. (X -> X) means consumed and produced types are the same, but (Any -> Any) might not.

multi-apply

--- Many students checked the case where the list is empty; this is unnecessary, since foldl/foldr handles that implicitly.
**aop**

Many students applied `expt` outside of `build-list` and used a redundant `lambda` to create the list. It is better to use `expt` inside `build-list` to avoid redundancy.

**multi-compose**

Again, many students checked the case where the list is empty, which `foldl/foldr` handles implicitly.