Design Recipe

Purpose

— Helper functions need a purpose (unless otherwise specified).
— Do not forget to include parameters in the header of a purpose.

Contract

— Make sure to use brackets around listof ... in contracts.
— Types should have their first letters capitalized.
— Write Str for strings, Bool for booleans, and (listof ...) for lists. Note the abbreviation and capitalization.
— Helper functions need contracts.
— Many students forgot to capitalize both letters of EV in their input types—note that this is how we defined it in the data definition.

Examples

— Helper functions need at least one example.

Question 1

in-range

— Do not create a list as an accumulator and then take the length of the list. This is unnecessary, and some students used append to accomplish this, which is worse.
— Remember that (< b a) is possible.

spread

— Setting the minimum to 0 for spread/acc is not correct. Since the list is non-empty, you can have the first element be the initial minimum.
— Remember you can have multiple accumulators. Many students did well but there were some complex solutions. You can have one accumulator tracking the maximum, and the other tracking the minimum.

sel-sort

— Forgetting empty case for sel-sort. Remember to write your contracts.
— When smallest-first calls smallest-first/acc, you should pass (rest lon) as the list to recurse through to ensure the length does not change. This works because we also pass the first element as the current minimum so we don’t lose any elements of the list.
— In smallest-first/acc, you need to ensure no elements are lost for each recursive call. In case you find a new minimum, the first of the list becomes the new minimum, and the previous minimum gets added to the list accumulator.
Question 2

Data definition

— ev should be a structure, not a list.

adjust-prices

— Be specific enough about the parameter percent-change in the purpose: simply mentioning that the prices are adjusted is not enough.

— Use (make-ev ...) to create a new EV. Remember that an EV is not a list, so use a selector function to access the price.

build-inventory

— Simply copying “produces a list of EVs with the same information” from the assignment does not make for a valid purpose.

— year, milage, and mpg should all be lists of natural numbers, and do not forget to include the requirement that all input lists must be of equal length. Remember to format your lists correctly: (listof Nat).

— Use accumulative recursion to avoid complicated code.

compare-ev

— Remember to talk explicitly about what the function produces or any sort of comparison (i.e. there was no talk about less than, equal to, greater than, or the symbols themselves).

— This question was done well. However, a common error was producing the incorrect symbol when comparing ev-mileage.

sort-evs

— Since the most common ways to solve the question were to use the question 1 (c) code and to use insertion sort from the slides, the usage of append resulted in a loss of marks as neither of these approaches needed append.

— Do not over-complicate things by creating maximum and minimum functions (when it was not needed). Use compare-ev as a helper function to simplify the code!

Question 3

— Overall, both parts of this question were well done.

— The correct type in the contract was (setof Nat). Students did not need to specify any requirements under the contract: we already know that sets are ordered and have no duplicates.

— Students needed to check (= (first s1) (first s2)) to avoid repeating elements.

— Some students used append, member?-like functions, or accumulative recursion in their solutions. append is complicated under the hood, students were instructed not to use member?, and approaches via accumulative recursion were very complicated. See the posted solutions.