Post-Mortem
Assignment 02
January 29, 2018

We normally publish the post-mortem for an assignment after it has been marked and released. Here is a list of common errors provided by the graders for assignment 2.

Style and Spacing

• Constants (and helper functions) should be defined above the design recipe for the function they are used in. Many students defined their constants and helper functions between their examples and function definition, or after the main functions specified in the assignment.

• Helper functions require their own purpose, contract, and examples.

• Lines should be less than 80 characters long: excessively long lines should be broken up into multiple shorter lines, and make use of DrRacket’s auto-indenting features.

• Some students did not lay out their cond question/answer pairs consistently, and had a varying amount of question/answer pairs on each line.

• Some students forgot to leave a blank line before and after function definitions, or included both the function header and part of the function body on the same line.

General

• Remember to check basic test results after submitting to catch simple (but impactful) errors.

• Many students used parameter names that were not meaningful (e.g., a, b, c) or used ambiguous/unclear names (e.g., pm1, s11, we). It is better to have names that are long and clear than names that are short and cryptic.

• The names of helper functions should also be clear and descriptive. For example, helper, test, input, function, and compute are very poor choices for helper function names.

• For now, equal? should only be used to compare two values of unknown types, or values that can take on more than one type. When the types of the arguments are known, use the most appropriate comparison function (e.g., symbol=?).

• To determine if a boolean value is true or false, use that value directly. Expressions like (boolean=? param true) or (equal? param false) are overly complex.

• Starting a new cond expression in an else clause is unneeded. Instead, directly check for the next condition in the original cond expression.

• Symbol values should not be defined as constants. As stated in the course notes, they are self-documenting, and defining them as constants does not provide any additional meaning.
• As specified in the assignment preamble, any symbol values should exactly match the description in the questions. Some students used symbols that differed slightly from those written in the assignment (such as using ‘gray instead of ‘grey, or ‘highschool instead of ‘high-school), and thus lost some correctness marks.

Design Recipe

• Purposes, contracts, and function definitions should not be explicitly labelled.
• Purposes should begin with a function header (e.g., (func-name param1 param2 param3)). These headers should include each of the parameter names used in the function.
• Purposes should meaningfully use each parameter name in the description of the function, and these references to the parameter names should be written exactly as they appear in the function header.
• Contracts should begin with func-name:.
• If restrictions are already implied in a data type, they do not need to be included in the requires: section. For example, if a function consumes a Nat, there is no need to specify that the consumed Nat should be greater than or equal to 0.
• Requirements are not needed for the values and types that functions produce.

Question 1

• Many students did not use a helper function to convert the colours into their associated values.
• Many students defined three helper functions to convert each of the colours into their associated values. However, the definition of these three helper functions were often exactly the same, which defeats the purpose of writing helper functions to reduce duplicate code. Please review the scoping rules in Module 1 to realize that only one helper function is needed for a task like this.
• Some students did not specify that the consumed symbols for compute-resistance could only be one of the 10 colours specified on the assignment.

Question 2

• Many students did not define constants for the range thresholds associated with each parameter, and the point values awarded for the respective ranges.
• In part (a), many students did not make use of any helper functions, and had a large block of code for their solution in immigration-points. Although helper functions do not decrease code duplication in this question, they should be used for clarity, and to break down the problem into smaller parts.
• Some students did not make use of patterns when computing the points associated with each age, and enumerated all the possibilities from the ages 35 to 46 in their solution.
• Some students did not correctly specify that immigration-points produces a Nat. Remember that contracts should always use the most specific types possible.
• In part (b), some students did not use immigration-points as a helper function, which lead to unnecessary code duplication.
• Many students did not include a test case for `eligible-skilled-worker` where the value produced from calling `immigration-points` would be exactly 67. This test should be included, as 67 is considered a break point in the context of this question.

• For both parts (a), and (b), many students did not include requirements for the primary language level, the secondary language level, and the education level.

**Question 3**

• Many students did not define constants for the expiry date thresholds of 30 and 120.

• Purposes should only describe what the function does and not how it does it (i.e., they should not specify that the function was implemented using only cond/boolean expressions).

• In part (b), many students struggled with grouping the boolean expressions together, and were either trying to combine the boolean expressions with an `or` on the outside, or they did not make use of the `not` function to handle special cases (such as the ‘dairy and ‘meat cases).