Design Recipe - Common Errors

Purpose:

- Remember to reference parameters in your purpose statement.

Contract:

- Ensure data types are as specific as possible. For example, using "Int" over "Nat", where it is appropriate.
- Do not use full forms for data types. For instance, use "Sym" instead of "Symbol" as specified in the Style Guide
- Remember to completely consider the valid input conditions defined in the question specification before writing your contract. This way you can see what your contract misses after writing it and implement that into your "Requires" section.
  - Try to be as specific as you can with your contract before appending a "Requires" section

Naming:

- Avoid ambiguous parameter names.

Examples:

- Remember to make your own examples and try to not copy results from the assignment pdf to avoid any errors with our system. You can double check your submission on Markus in plain text to see if there are any additional characters from the pdf.

Question 2 (Resistance)

compute-resistance:

- Use constants to eliminate magic numbers in your code.

Question 3 (Immigration)

immigration-points:

- Remember to add requirements for parameters as specified in the problem description.
- When calculating the points for age, you should not have used a separate conditional for each age from 36 to 45. Here is an example of a good helper that calculates the points for age:

```
;; (age-points age) produces the number of points associated with the
;;  given age.
;; age-points: Nat -> Nat
;; Examples:
(check-expect (age-points 0) 0)
(check-expect (age-points 40) 7)
(check-expect (age-points 55) 0)
(define (age-points age)
  (cond
    [(<= age age-threshold-a) age-a]
    [(<= age age-threshold-b) age-b]
    [else (max 0 (+ 36 (- age-c-base age)))]))
```

Question 4 (edible-cond)

still-edible/cond?:

- A common mistake was missing constants for the benchmark days past best before date.