CS 131 - DESIGN RECIPE - VERSION 1

You are expected to follow the Design Recipe for all functions that you write.

Because the Design Recipe is so important, you will receive **significant** credit for the signature, purpose, header, and examples/check-expects portions of your functions (and later for the data definitions and body templates as well). Typically you'll get at least half-credit for a correct signature, purpose, header, and examples/check-expects, even if your function body is not correct (and, you'll **lose at least half-credit** if you omit these or do them poorly, even if your function body is correct).

So, designing each function is expected to include:

Step 1 - problem analysis and data definition

- Consider your problem; consider the kinds of data involved in your problem. Determine if you need to define any new kinds of data, and develop data definitions to do so as applicable. We will be discussing this more as the semester proceeds.

Step 2 - signature/purpose/header

- First develop a signature comment, including a nicely-descriptive name for your function, the **types** of expressions it expects, and the **type** of expression is produces. This should be written as discussed in class (and you can find examples posted on the public course web page). For example,

```
; signature: rect-area: number number -> number
```

- Then develop a purpose comment, **describing** what the function expects and **describing** what it produces. For example,

```
; purpose: expects the length and width of a rectangle,
;          and produces the area of that rectangle
```

- Now write the function header, giving a good, descriptive name for each parameter variable. Use `...` as a stub for the function body at this point.

Step 3 - develop specific examples/tests

- Now develop `check-expect` (or `check-within`, or other `check-` operation) expressions expressing specific examples of your function that you devise **before** writing your function body.
  - (These may be **placed** before or after your actual function definition, but you are expected to **create** these **before** writing the function body. I'll have no way of knowing if you really write these in the correct order, but note that I won't answer questions about your function body without seeing your examples written as `check-expect` (or appropriate `check-` operation) expressions first...)
  - For example,

```
(check-expect (rect-area 3 4)
              12)
```

- How many check-expect expressions should you have? That is an excellent question, and a major course topic.
  - We'll be discussing how you determine how many you need, and later you'll be graded based on whether you include a reasonable number and kind of `check-` expressions.
  - The **basic rule of thumb** is that you need a specific example/`check-` expression for each "case" or category of data that may occur... and you can always add more if you'd like!

Step 4 - decide which body template is appropriate

- Replace the `...` that is currently your function body with an appropriate template, based on the problem type. We will be discussing this as the semester progresses.

Step 5 - Develop/complete the function's body

- Either replace the `...` that is currently your function body, or finish filling in/completing the body template you developed in Step 4.

Step 6 - Run the tests

- Click the Run button! 8-)
- Note that you may include as many additional calls or tests of your function as you would like after its definition.