CIS 130 - Homework #2

Due:

Thursday, February 11th, 11:59 pm

Purpose:

To provide practice giving a name to an inserted image, writing functions using the design recipe, using named constants, and writing simple programs that call functions.

How to submit:

When you are done with the following problems:

- save your resulting Definitions window contents in a file with the suffix `.ss` or `.scm`
- transfer/save that file to a directory on **nrs-labs**
- use **ssh** to connect to **nrs-labs**
- cd to the directory/folder where you saved it (`cd 130hw2` for example)
- use `~st10/130submit` to submit it
- make sure that `~st10/130submit` shows that it submitted your homework `.ss` or `.scm` file
- (ASK ME if this is not clear, or if you have any problems with submission!)

Important notes:

- Each student should work individually on this assignment.
- You are expected to follow the Design Recipe for all functions that you write. So, each function is expected to include:
    - a contract comment, including the name of the 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,

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

    - 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
      ```

    - [following the design recipe, you will be writing the function header next; note that you don't need to write it twice. Follow the function header with a body of ... at this stage, and replace that ... with its body later, at the appropriate step in the design recipe.]
    - check-expect expressions expressing the specific examples you devise **before** writing your function body. (These may be **placed** before or after your actual function, 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 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 topic.
- For this homework, I'll say how many you need, but 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-expect expressions.
- The basic rule of thumb is that you need an example/check-expect for each "case" or category of data that may occur... and you can always add more if you'd like!

- [and, of course, your function definition itself!]
- You may include as many additional calls or tests of your function as you would like after its definition.

- **Because the Design Recipe is so important,** you will receive **significant credit** for the contract, purpose, header, and examples/check-expects portions of your functions. Typically you'll get at least half-credit for a correct contract, 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).

Homework problems:

Problem 0

For this assignment, you should be using the **Beginning Student** language in DrScheme, with the **universe.ss** and **fabric-teachpack.scm** teachpacks installed.

In the definitions window, type in a comment-line containing your name, followed by a comment-line containing CIS 130 - HW 2, followed by a comment-line with no other text in it --- that is,

```
; type in YOUR name
; CIS 130 - HW 2
;
```

Problem 1

Below what you typed in Problem 0 above, type the comment lines:

```
; Problem 1
;
```

Obtain an image -- of one of the formats .jpg, .png, or .gif -- no larger than 100 pixels by 100 pixels. (You can find one on the Web, or create it in some application, etc.) Write a Scheme definition giving a descriptive name to this image (inserting the image into your DrScheme definitions window).

Then, write an expression that uses `add-print` with this name you've defined to add this image atop another image of your choice.

Problem 2

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 2
;
```

Using the design recipe, design a function that produces the volume of a rectangular tank. (You'll need to consider: how many and what type of expressions should such a function expect, to be able to produce such a volume?)

One example should suffice for this function.

Problem 3

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 3
;
```

Using the design recipe, design a function that produces the average gas consumption (in miles-per-gallon) used for a trip. (You'll need to consider: how many and what type of expressions should such a function expect, to be able to produce this average?)

One example should suffice for this function.

Problem 4

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 4
;
```

Using the design recipe, design a function that expects a desired length in pixels and produces a red square image whose sides are each that length.

Provide at least two examples for this function.

Problem 5

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 5
;
```

Write a Scheme definition that will define the name `MIN-PER-HR` to be the number of minutes in an hour. (This name is an example of a **named constant**.)

Then, using the design recipe and this named constant, design a function `minutes->hours` that expects a number of minutes and produces the number of hours equivalent to that number of minutes.

(Yes, that "arrow", the dash and right angle bracket `->`, is intended -- it is not a typo, and it is permitted in a Scheme identifier.)

Provide at least two examples, at least one for a number of minutes less than 60, and at least one for a number of minutes greater than 60.

Problem 6

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 6
;
```

Using the design recipe, design a function `minutes->wages` that expects the number of minutes someone has worked and his/her hourly wage, and produces the amount he/she is owed (before taxes or any other adjustments). For full credit, you need to **use** the function `minutes->hours` within the body of `minutes->wages`.

Provide at least two examples, at least one for a number of minutes less than 60, and at least one for a number of minutes greater than 60.

Problem 7

Skip a line, and write a comment noting that what follows are your expressions for:

```
; Problem 7
;
```

Write a Scheme definition, another named constant, that will define the name `IN-PER-FT` to be the number of inches in a foot.

Then, using the design recipe and this named constant, design a function `total-inches` that expects a number of feet and a number of inches, and produces the total number of inches. (For example, the value of the expression `(total-inches 4 5)` should be `53`, because 4 feet and 5 inches is 53 inches overall.)

Provide at least two examples, at least one for a number of feet of 0, and at least one for a number of feet greater than 1.

Problem 8

Consider your named image from Problem 1. Also consider the `overlay` and `circle` operations you used in Problem 6 on HW #1.

Using the design recipe, design a function `frame-it` that expects a desired "matte" color, a desired "frame" color, and a desired length in pixels. Then it produces the following image: your named image from Problem 1 atop a circle of the desired "matte" color with that desired length as its diameter, atop a rectangle of the desired "frame" color with that desired length as its width and height.

Provide at least two examples.