CIS 130 - Homework #4

Due:

Thursday, March 4th, 11:59 pm

Purpose:

Practice designing auxiliary functions, boolean functions/predicates, and functions using a `cond` expression. Also more practice writing functions that use auxiliary 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 130hw4` 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.
- Remember, you are expected to follow the Design Recipe for all functions that you write.
- Remember to follow the class style guidelines (such as those listed in Homework 3).
- Now that we are writing conditional functions, be especially careful to follow the guidelines as discussed in class for the number and kinds of tests needed for functions (at least one for each "category" of data, plus, when applicable, at least one test for each "boundary" between such categories).

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 4, followed by a comment-line with no other text in it --- that is,

```
; type in YOUR name
; CIS 130 - HW 4
```

```
;
```

Problem 1

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

```
; Problem 1
;
```

(Modified version of Exercise 4.2.1 from HtDP 1st edition, pp. 33-34)

A function that produces a value of type boolean is sometimes called a boolean function, and sometimes it is also called a predicate.

Write a separate boolean function/predicate for each of the following intervals, that expects a number, and produces the boolean value `true` if the number is part of the interval described, and that produces the boolean value `false` if the number is not part of the interval described.

Note that:

- for this particular problem, the use of named constants for these particular intervals' end points is not required.
- a major part of this problem is developing enough specific examples/test cases (including all cases and boundaries between cases).
- hint: you should not need to use `cond` for these; it is not incorrect if you do, but it is better if you do not. The boolean operations `and`, `or` and `not` can be very useful here...

1 part a

Write the boolean function `is-within` to test if a number is within the interval (3, 7]

1 part b

Write the boolean function `in-either` to test if a number is in the union of (1, 3) and (9, 11). (For example, 2 is in this union; so is 9.5. But 3.7 is not...)

1 part c

Write the boolean function `is-outside` to test if a number is in the range of numbers **outside** of [1, 3].

Problem 2

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

```
; Problem 2
;
```

Imagine a program that, amongst its other tasks, greets its user in a manner appropriate to the current time of day. We might like a small function that expects the current hour of the day written in military time (0 is midnight, 1 is 1 am, ... 12 is 12 noon, 13 is 1 pm, ... 23 is 11 pm). Then, it could produce a greeting appropriate to the hour of the day.

So, write such a function. That is, write a function `get-greeting` that expects an hour of the day written in military time as described above, and produces a greeting appropriate to the morning if the hour is in [0, 12), it produces a greeting appropriate to the afternoon if the hour is in [12, 18), and it produces a greeting appropriate to the evening if the hour is in [18, 23].

What if this ever gets called with a number not in [0, 23]? That shouldn't happen, but if it does, something is seriously awry -- produce a greeting or a message indicating confusion in that case.

Problem 3

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

```
; Problem 3
;
```

Note: you can compare whether two expressions of any type have the same value by using the `equal?` function. (= only works for comparing numbers...)

```
(equal? "hi" "hi")
```

...has the value `true`, while:

```
(equal? "hi" "lo")
```

...has the value false.

A store gives discounts to frequent shoppers based on their past level of purchases; they are either "silver" level, "gold" level, or "platinum" level. Silver level frequent shoppers receive a 10% discount, gold level frequent shoppers receive a 25% discount, and platinum level frequent shoppers receive a 33% discount. All other shoppers receive no discount.

Design a function that expects a string representing the level of frequent shopper and produces the appropriate discount for that level written as a decimal fraction. (It should produce a discount of 0 if the string is not one of those noted above.) Be especially careful to use named constants as appropriate.

Problem 4

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

```
; Problem 4
;
```

Design a function that expects a string representing the level of frequent shopper and the total of a purchase before discount, and produces the appropriate discounted total for that purchase. For full credit, your solution must appropriately use your function from Problem 3.

Problem 5

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

```
; Problem 5
;
```

(Adapted from Keith Cooper's section of Rice University's COMP 210, Spring 2002)

Conditionals (and Pizza Economics)

This problem considers an important consequence of increased pizza consumption —the need for additional exercise.

Develop a function `workout` that determines the number of hours of exercise required to counter the excess fat from eating pizza. `workout` expects a number that represents daily pizza consumption, in slices, and produces a number, in hours, that represents the amount of exercise time that you need.

For a daily intake of :	You need to work out for :
0 slices	1/2 hour
1 to 3 slices	1 hour
>3 slices	1 hour +1/2 hour per slice above 3

Problem 6

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

```
; Problem 6
;
```

Here is another boolean function/predicate, which will be an auxiliary function for a later problem. We're going to work our way toward an animation in which something moves for a while, and then stops for a while, and then moves for a while, and then stops for a while.

As a first step, we decide it would be useful to have a function that simply tells us if the modulo of a time counter and the scene's width is less than some constant value that you choose.

6 part a

To start, define named constants for a scene's width and height, and for a backdrop of your choice.

Then, consider -- how long would you like for an object to be moving, before it stops? Decide, and define a named constant for this MOVING-MAX value. (Make sure this MOVING-MAX value is not bigger than your scene's width and height.)

6 part b

Now write a boolean function/predicate `in-moving-time` that expects a time-counter value, and produces whether or not the modulo of that time-counter value and the scene's width is in the interval `[0, MOVING-MAX]`.

Problem 7

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

```
; Problem 7
;
```

In this problem, you are just going to set up named constants related to the "moving parts" of the upcoming animation.

Now, consider: what do you want to have moving, for this homework's animation? You must at least meet the following requirements:

- some image needs to shown in a position based on the time-counter value when `in-moving-time` is true for that time-counter value;
- some image needs to be shown in the SAME position (that is, with the same x and the same y values) when `in-moving-time` is false for that time-counter value.

Now, these can be the same image, or different images. The image can simply be a static image, or it can be an image that itself changes based on the time-counter value (its size or color could change based on the time-counter value, for example).

If you are simply going to have one (or two) unchanging static image(s), define named constant(s) for them here.

If you are going to have a changing image, then declare any named constants for aspects of that image that are not going to change, if any. (For example, if a rectangle's width will change, but not its height, you would declare a named constant for that unchanging height. Or if you are going to change the blue and green values for an image's color, but not its red value, then you would declare a named constant for that unchanging red value.)

And if there are no named constants applicable for your moving image(s), then simply put a comment saying so as your answer to Problem 7.

Problem 8

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

```
; Problem 8
;
```

Now, write a function `create-cond-scene` that expects a time-counter value, and produces a scene meeting at least the minimum criteria already mentioned above, but repeated here too:

- some image within the scene needs to shown in a position based on the time-counter value when `in-moving-time` is true for that time-counter value;
- some image within the scene needs to be shown in the SAME position (that is, with the same x and the same y value) when `in-moving-time` is false for that time-counter value.

Your function must use the `in-moving-time` function from Problem 6, of course. It should also use your named constants (if any) from Problem 7. And, this function should be followed by a call to the `animate` function using this function.

Beyond the above, you may be as elaborate or as simple as you wish. If applicable, define additional named constants for use by your function. When `in-moving-time` is true for that time-counter value, you get to decide how the time-counter value affects the image's position; when it isn't, you get to decide if anything else about the image is affected by the time-counter value (since its position should not be affected by the time-counter value in that case). But, it is OK if some unchanging image is just shown in the same location in that case.

If your `create-cond-scene` would be easier if you defined some additional auxiliary functions, feel free to add these -- just be sure, of course, to follow the design recipe for each such auxiliary function.

There is a small chance that I might be able to post a few examples of these on the public course web page at some point in the future -- just in case this happens, please let me know in a comment if you would want your name to accompany your scene.