Assignment: 2

Due:	Tuesday, January 23, 9:00 pm
Language level:	Beginning Student
Files to submit:	`resistance.rkt`, `immigration.rkt`, `fridge-foods.rkt`, `bonus.rkt`
Warmup exercises:	HtDP 4.1.1, 4.1.2, 4.3.1, 4.3.2
Practice exercises:	HtDP 4.4.1, 4.4.3, 5.1.4

- Policies from Assignment 1 carry forward.
- This assignment covers concepts up to the end of Module 02. Unless otherwise specified, you may only use Racket language features we have covered up to that point.
- Make sure you read the **OFFICIAL A02 post on Piazza** for the answers to frequently asked questions.
- For this assignment we have provided starter files to help you organize your functions. You are not obligated to use them, but they reflect the style and organization we expect of your work. Download them from the assignment page.
- Your solutions must be entirely your own work.
- Solutions will be marked for both correctness and good style. Follow the guidelines in the Style Guide.
- **Unless otherwise specified, for this and all subsequent assignments, you should include the design recipe as discussed in class.**
- The basic and correctness tests for all questions will always meet the stated assumptions for consumed values.
- You must use *check-expect* for both examples and tests.
- You must use the **cond** special form, and are not allowed to use **if** in any of your solutions.
- **It is very important that your function names match ours.** Use the basic tests to check this. In most cases, solutions that do not pass the basic tests will not receive any correctness marks. The names of the functions must be written exactly. The names of the parameters are up to you, but should be meaningful. The order and meaning of the parameters are carefully specified in each problem.
- Any string or symbol constant values must exactly match the description in the questions. Any discrepancies in your solutions may lead to a severe loss of correctness marks. Basic tests results will catch many, but not necessarily all of these types of errors.
- Since each file you submit may contain more than one function, it is very important that the code runs. If your code does not run, then none of the functions in that file can be tested for correctness.

- Do not send any code files by email to your instructors or ISAs. Course staff will not accept it as an assignment submission. Course staff will not debug code emailed to them.
- You may use examples from the problem description in your own solutions.

Here are the assignment questions you need to submit.

1. You may be familiar with the electronics component called the *resistor*. Each resistor has a *resistance*, measured in Ohms (Ω). For example, a particular resistor may have a resistance of 4700 Ω. Historically, resistances were indicated with a series of coloured bands. Most resistors have four or five such coloured bands; the first three are used to indicate the resistance in the following way:

   - The resistance is written as a two-digit number, multiplied by a power of ten. For example, $4700 = 47 \cdot 10^2$. Write down the two digits of the number, along with the exponent. In this example, we would have $(4, 7, 2)$.
   - Convert the three numbers (in order) to colours according to the following chart:

0	`'black`	1	`'brown`	2	`'red`	3	`'orange`	4	`'yellow`
5	`'green`	6	`'blue`	7	`'violet`	8	`'grey`	9	`'white`

     So our example resistor would have its first three bands coloured yellow, violet, and red (in that order).

   Write a Racket function `compute-resistance` which consumes three colours (as symbols, such as `'red`, `'blue`, `'grey`, etc.) representing the first three colours of the resistor bands. It should produce the resistance of the resistor, in Ohms. Continuing with our example, `(compute-resistance 'yellow 'violet 'red)` produces 4700.

   Place your function in the file `resistance.rkt`.

2. As part of its immigration policy, Canada uses a "Comprehensive Ranking System" to assess whether applicants qualify for the "skilled worker" category. In this question we will implement a simplified version of this system [1]. Place your solutions in `immigration.rkt` .

   The ranking system produces a score out of 100 points. To be considered for the skilled worker category, applicants must score 67 points or higher.

   (a) First, write a function `immigration-points` which produces the number of points an applicant receives. The function consumes the following parameters (in the given order):

[1] This is based on the data from https://www.canada.ca/en/immigration-refugees-citizenship/services/immigrate-canada/express-entry/become-candidate/eligibility/federal-skilled-workers/six-selection-factors-federal-skilled-workers.html , as retrieved on 2017-12-27.

- `primary-language-level` which is a natural number between 1 and 9 inclusive. A value of 9 contributes 24 points; a value of 8 contributes 20 points, a value of 7 contributes 16 points, and any other value contributes zero points.
- `secondary-language-level` which is a natural number between 1 and 9 inclusive. A value of 5 or higher contributes 4 points, and any other value contributes zero points.
- `education-level` which takes one of the following values: `'postgraduate` which contributes 25 points, `'undergrad` which contributes 19 points, `'high-school` which contributes 5 points, or `'no-secondary` which contributes zero points.
- `work-experience` which is a natural number indicating the number of years of work experience the applicant has. One year contributes 9 points, 2-3 years contribute 11 points, 4-5 years contribute 13 points, and 6 or more years contribute 15 points. Zero years contributes zero points.
- `age` which is a natural number indicating the person's age. An age 17 or under contributes 0 points, 18-35 inclusive contributes 12 points, 36 contributes 11 points, 37 contributes 10 points, and so on until age 46 which contributes 1 point. Age 47 or higher contribute zero points.
- `job-offer?` which is a Boolean indicating whether the applicant has a job offer in Canada. `job-offer?` is `true` if the applicant has a job offer, and `false` otherwise. A job offer contributes 10 points, and no job offer contributes zero points.
- `adaptability` which is a natural number indicating a number of other adaptability factors that apply (which we do not detail in this question). Two or more adaptability factors contribute 10 points, one adaptability factor contributes 5 points, and no adaptability factors contribute zero points.

You will likely find helper functions useful in writing this function. Be clever about reducing the number of conditions you have to evaluate for each criterion (for example, evaluating the `age` criterion can be done in no more than three conditions).

(b) Write a second function `eligible-skilled-worker?` which produces `true` if the given applicant is eligible for the skilled worker category, and `false` otherwise. This function consumes the same parameters as `immigration-points`, and takes into account the number of points and the following restriction: regardless of point total, any applicant who has a primary language level lower than 7 is immediately disqualified from the skilled worker category.

For example, `(immigration-points 8 3 'undergrad 1 23 false 1)` produces `65`.

`(eligible-skilled-worker? 8 3 'undergrad 1 23 false 1)` produces `false`.

You do not have to check for applications that are impossible in combination so long as individual arguments obey their requirements. For example, we will not test cases where the applicant is 8 years old but has 10 years of work experience.

Notes about parameters: these functions require you to write functions with many parameters. To avoid lines being longer than 80 characters (and thus losing style marks) your function headers can be broken up over more than one line.

Notes about constants: somewhat unfortunately, this question will require you to define a large number of constants. Some guidelines:

- Constants for ranges of values should be named consistently. For example, naming one constant `education-highschool` and a closely related one `postgrad-points` is probably a bad idea. Using `education-highschool` and `education-postgraduate` would be more consistent. Choose a consistent naming scheme and stick with it.
- To the extent possible, you should avoid including fixed values in your constant names. For example, `secondary-language-4-points-level-5` is probably a bad name, because changing the point value (or the language level necessary to earn those points) means changing the name of the constant.

  Using the terminology "range" instead of hardcoding numbers in your names is one way around this problem. (There are others!) For example, `secondary-language-range-a` might indicate the 5 point level, and `secondary-language-range-b` might indicate the 0 point level.
- Related constants should be grouped together in your code. One line of whitespace should separate each group of constants.

Notes about testing: Testing code with this many conditions can be a challenge. Here are some guidelines that might make your life easier:

- Start by finding a combination of parameters that produces zero points.
- Using this initial test as a base case, vary other parameters in isolation to vary the point totals for each condition.
- When testing intervals, be sure to test both boundary conditions and conditions within the range.
- Make sure you have sufficient tests that every line of code is exercised. There should be no "black highlighting."

3. You have been so busy studying that you have neglected to buy groceries. Now you are hungry and looking through your fridge. There are some foods inside but you are not sure whether they are safe to eat, so you write a Racket function to help you decide[2]:

   - If the item is mouldy then it is not safe to eat.
   - Otherwise, if the item is of type `'dairy` or `'meat` and the item is 30 or more days past its best before date, then it is not safe to eat.

[2] What?! This is bad life advice.

- Otherwise, if the item is of type `'condiment` it is safe to eat.
- Otherwise, if the item is 120 or more days past its best before date, then it is not safe to eat.
- Otherwise, the item is safe to eat.

(a) Write a function `safe-to-eat/cond?` which consumes a Boolean parameter `mouldy?`, a symbol `food-type`, and a natural number `days-past-best-before` and produces `true` if the food is safe to eat and `false` otherwise. The function should use **`cond`** and no Boolean operations (so no **`and`**, **`or`** or `not`).

Carefully consider the order of your conditions to minimize duplicated code.

For example, `(safe-to-eat/cond? false 'condiment 200)` produces `true`.

Similarly, `(safe-to-eat/cond? true 'beans 0)` produces `false`.

You may not use additional helper functions in this question. You do not need to check that `food-type` is actually a valid type of food.

(b) Write a Racket function `safe-to-eat/bool?` which consumes the same parameters as `safe-to-eat/cond?` but uses Boolean operations only, and no instances of **`cond`**.

Place these functions into the file `fridge-foods.rkt`

This concludes the list of questions for which you need to submit solutions. Don't forget to always check your email for the basic test results after making a submission.

Bonus (5%): Write a function `date->day-of-week` which consumes a natural number and produces a symbol corresponding to the day of the week, according to the Gregorian calendar.

An integer encodes a date as follows:

- The final two digits correspond to the day of the month
- The two digits before that correspond to the month of the year
- The digits before that correspond to the year

The function should return one of eight symbols: `'Invalid` if the consumed number does not correspond to a valid date, or one of `'Monday`, `'Tuesday`, `'Wednesday`, `'Thursday`, `'Friday`, `'Saturday`, `'Sunday` according to the day of the week the consumed date corresponds to.

Here are some examples:

```
(check-expect (date->day-of-week 38781202) 'Monday)
(check-expect (date->day-of-week 18321401) 'Invalid)
(check-expect (date->day-of-week 1920303) 'Invalid)
```

Any date before October 1752 should be considered `'Invalid` (even though this is not technically true).

You may only use the Racket constructs we have discussed in lecture so far, and built-in mathematical functions. You may not use Racket's date functions.

It is acceptable to consult rules or algorithms for computing the day of the week given a date, but it is not acceptable to copy and paste code you have found. You must cite any resources you use in your Racket file.

Put your solution in `bonus.rkt`.

Challenges and Enhancements: *Reminder—enhancements are for your interest and are not to be handed in.*

check-expect has two features that make it unusual:

1. It can appear before the definition of a function it calls (this involves a lot of sophistication to pull off).
2. It displays a window listing tests that failed.

However, otherwise it is a conceptually simple function. It consumes two values and indicates whether they are the same or not. Try writing your own version named *my-check-expect* that consumes two values and produces 'Passed if the values are equal and 'Failed otherwise. Test your function with combinations of values you know about so far: numbers (except for inexact numbers; see below), booleans, symbols, and strings.

Expecting two inexact numbers to be exactly the same isn't a good idea. For inexact numbers we use a function such as *check-within*. It consumes the value we want to test, the expected answer, and a tolerance. The test passes if the difference between the value and the expected answer is less than or equal to the tolerance and fails otherwise. Write *my-check-within* with this behaviour.

The third check function provided by DrRacket, *check-error*, verifies that a function gives the expected error message. For example, (*check-error* (/ 1 0) "/: division by zero")

Writing an equivalent to this is well beyond CS135 content. It requires defining a special form because (/ 1 0) can't be executed before calling *check-error*; it must be evaluated by *check-error* itself. Furthermore, an understanding of *exceptions* and how to handle them is required. You might take a look at exceptions in DrRacket's help desk.