Winter 2025
Clarke

Assignment: 03

Due:	Tuesday, February 4, 2025 9:00 pm
Coverage:	L06
Language level:	Beginning Student
Allowed recursion:	Second version of the rules
Files to submit:	`rgb.rkt`, `robot.rkt`, `lists.rkt`, `pnormp.rkt`, `bonus-a03.rkt`

Assignment policies:

- These policies apply to all assignments.
- Make sure you read the official assignment post on **ed**.
- You may not use functions or language constructs from lectures after the "coverage" lecture listed above.
- Functions and symbols must be named **exactly** as they are written in the assignment questions. You may define helper functions, if needed.
- You must provide a purpose, contract, and appropriate test cases for all required functions, i.e. those we explicitly ask you to write.

Here are the assignment questions you need to solve and submit.

1. **(24%)**: In computer graphics and image processing colour is typically represented as a triplet (R,G,B) of whole numbers representing the red, green, and blue components respectively. In most images, the values for each colour are in the range $[0,2^8-1]=[0,255]$.

   Humans, however, do not typically think of colours according to their RGB values but by their names. For example, red is $(255,0,0)$, and yellow is $(255,255,0)$.

   In this question, we'll store the RGB-triplet as a three-element natural number list. For example, `(cons 255 (cons 0 (cons 0 empty)))` is red.

   The data definition for an RGB triplet is as follows:

```
;; An RGB Triplet (RGB) is a (cons Nat (cons Nat (cons Nat empty)))
;; requires: each element in the list must be <= 255
```

   Place your solutions to the following in `rgb.rkt`.

   (a) Write a function called `RGB->name` that consumes an RGB-triplet and produces the colour name as a symbol. `RGB->name` should be able to identify: `'red`, `'green`, `'blue`, `'yellow`, `'cyan`, `'magenta`, `'white`, and `'black`. Symbol names must be in lowercase. For any other colour, the symbol `'unknown` should be produced. You may find the following table mapping RGB values to names helpful.

R	G	B	Name
255	0	0	red
0	255	0	green
0	0	255	blue
0	0	0	black
255	255	255	white
255	255	0	yellow
255	0	255	magenta
0	255	255	cyan

(b) Write a type predicate called `valid-rgb?` that consumes *anything* and produces `true` if the consumed argument is a valid RGB Triplet according to the data definition above and `false` otherwise. You may use the type predicate `integer?`.

Here are some example test cases:

```
(check-expect (valid-rgb? 'red) false)
(check-expect (valid-rgb? (cons 255 (cons 0 (cons 0 empty)))) true)
(check-expect (valid-rgb? (cons 314 (cons 159 (cons 26 empty))))
   false)
```

2. **(24%)**: A robot's state is given by its (x, y) position on an integer grid and the direction it's facing, one of `'North`, `'South`, `'East`, or `'West`. Due to power distribution requirements, robots are currently restricted to the square integer grid defined by the opposite corners $(0,0)$ and $(10,10)$ inclusive (that is, the robot may be at $(10,10)$). In this problem, we will be representing a robot's state as a three-element list storing its x coordinate, y coordinate, and direction (in that order). Place your solutions in `robot.rkt`.

   (a) Write a data definition for a robot's `State`.

   (b) Write a function `robot-ctl`. The robot control function consumes a `State` and a command, in that order. Commands are the symbols `'forward`, `'turn-left`, and `'turn-right`. `robot-ctl` produces a new `State`.

   A robot command of `'turn-left` or `'turn-right` always succeeds. The produced `State` will be the same as the consumed `State` except that the direction will be different. For example, a robot facing `'North` and told to `'turn-left` will then face `'West`. Additional `'turn-left` commands will cause it to face `'South`, `'East`, and finally, `'North` again.

   A robot command of `'forward` changes the y coordinate by 1 if facing `'North` and by -1 if it is facing `'South` and similarly for the x coordinate if it is facing `'East` or `'West`. However, if the robot is already at the edge of its power grid, the state does not change. The `'forward` command does not change the robot's direction.

3. **(36%)**: The following questions require recursion on lists. Remember to follow the rules. Place your solutions in `lists.rkt`.

   (a) Write a function `absolutely-odd` that consumes a list of integers and produces the sum of the absolute values of the odd integers in the list, or 0 if there are none.

```
(check-expect (absolutely-odd (cons -1 (cons 2 (cons 3 empty)))) 4)
(check-expect (absolutely-odd (cons 2 (cons 4 (cons 6 empty)))) 0)
(check-expect (absolutely-odd (cons 10 (cons 9 (cons 8 empty)))) 9)
```

   (b) Define a *spiraling list* as a list of integers with the following properties:

      i. The list alternates between positive and negative integers.
      ii. The absolute value of the integers strictly increases.

   A list of a single integer and the empty list are spiraling lists. Write a predicate `spiraling?` that determines if a list of integers is spiraling list.

```
(check-expect
  (spiraling? (cons 1 (cons -10 (cons 100 empty))))
  true)
(check-expect
  (spiraling? (cons -1 (cons 2 (cons -3 (cons 4 empty)))))
  true)
(check-expect
  (spiraling? (cons 99 (cons -100 (cons 100 empty))))
  false)
(check-expect
  (spiraling? (cons 0 (cons -10 (cons 100 empty))))
  false)
```

   (c) The *geometric mean* of n numbers is defined as the nth root of the product of those numbers. Thus, the geometric mean of the set $\{x_1, x_2, \cdots x_n\}$ is

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}.$$

   For example, the geometric mean of $\{9, 0.5, 6\}$ is 3. To avoid complex numbers, e.g., $\sqrt{-1}$, let's also assume the numbers are all positive ($x_1 > 0, x_2 > 0, \cdots$).

   Write a function `geometric-mean` that consumes a non-empty list of positive numbers and produces its geometric mean.

   Since the geometric mean may be an inexact number, e.g., $\sqrt{2}$ you should use `check-within` for this problem. A *delta* of 0.0001 should be acceptable for your `check-within` tests.

```
(check-within
  (geometric-mean (cons 9 (cons 0.5 (cons 6 empty))))
  3 0.0001)
(check-within
  (geometric-mean (cons 1 (cons 2 (cons 3 (cons 4 empty)))))
  2.2133 0.0001)
```

4. **(16%)**: Let p be any positive integer. The p-norm of a list of numbers $v = (v_1, v_2, ..., v_n)$ is denoted as $||v||_p$ and defined as

$$\|v\|_p = \sqrt[p]{\sum_{i=1}^{n} |v_i|^p}, \quad \text{and its } p\text{th power by} \quad \|v\|_p^p = \sum_{i=1}^{n} |v_i|^p.$$

   i.e., $\|v\|_p$ is the pth root of the sum of the pth powers of the absolute values of the entries of v. $\|v\|_p^p$ is similar, but conveniently doesn't involve taking a pth root. In later courses like linear algebra, scientific computation and machine learning, p-norms will be very useful!

   For example, with $p = 3$ and $v = (3, -4, 5)$ then

$$\begin{aligned} \|v\|_p = &\quad \|(3,-4,5)\|_3 = \sqrt[3]{|3|^3 + |-4|^3 + |5|^3} = \sqrt[3]{27+64+125} = \sqrt[3]{216} \\ \|v\|_p^p = &\quad \|(3,-4,5)\|_3 = |3|^3 + |-4|^3 + |5|^3 = 27+64+125 = 216 \end{aligned}$$

   Write a function `pnormp` which consumes a positive integer p and a list of numbers v, in that order, and produces $\|v\|_p^p$. The pth power of the p-norm of an empty list is 0.

   A *delta* of 0.0001 should be acceptable for `check-within` tests.

   Please put your function into a file called `pnormp.rkt`.

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

Assignments will sometimes have additional questions that you may submit for bonus marks.

5. **10% Bonus**: A rectangle is specified by a four element list as: `(cons xmin (cons xmax (cons ymin (cons ymax empty))))`.

   Write a function called `overlap-area` that consumes two rectangles and produces the size of the overlapping region. If the rectangles do not overlap, then 0 should be produced. A negative value should never be produced.

   Note: for all inputs you may assume that *xmin* < *xmax* and *ymin* < *ymax*. Place your solution in `bonus-a03.rkt`.