Assignment: 05
Due: Tuesday, February 23rd at 11:45 am Eastern Time
Language level: Beginning Student
Files to submit: reclists.rkt exit.rkt nutrition.rkt rainbow.rkt

• Join the discussion for the answers to frequently asked questions.
• Unless stated otherwise, all policies from the previous assignment carry forward.
• This assignment covers material up to the end of Module 6.
• The only built-in functions and special forms you may use are listed below. If a built-in function or special form is not in the following list, you may not use it:
  * + - ... / < <= = > >= abs and boolean? ceiling char<=? char=? char//=? char/>?
  expt first floor integer? list->string list? max member? min not number->string
  number? odd? or quotient remainder rest second sqr sqrt string->list string-append
  string-length string<=? string<? string=? string>=? string>? string? substring
  symbol=? symbol?
• Remember that we have temporarily removed the following functions from our toolbox:
  filter foldl foldr lambda length list map range
• Remember that basic tests are meant as sanity checks only; by design, passing them should not be taken as any indication that your code is correct, only that it has the right form.
• Unless the question specifically says otherwise, you are always permitted to write helper functions to perform any task. You may use any constants or functions from any part of a question in any other part.
• For any inexact tests, use a tolerance of 0.0001.

1. Working with lists recursively. Submit your solutions for all parts of this question in the file reclists.rkt

1.1. Summing lists.

Write a function sum-lists that consumes two (listof Num) of equal length and produces a list which is the pairwise sum of the elements. For example,

\[
\begin{align*}
&\text{(check-expect (sum-lists (cons 17 (cons 32 (cons 51 empty))) (cons 5 (cons 0 (cons 19 empty)))))} \\
&\quad (\text{(cons 22 (cons 32 (cons 70 empty))))) \\
&\text{(check-expect (sum-lists (cons 17 (cons 32 empty)) (cons 0 (cons 0 empty))})} \\
&\quad (\text{(cons 17 (cons 32 empty))})
\end{align*}
\]
1.2. Zip.

Recall the function `zip` you wrote in A03. Complete `zip` again, this time however within the restrictions of this assignment. For example,

```
(check-expect (zip (cons "McDavid" (cons "Draisatl" (cons "Kassian" empty)))
  (cons 97 (cons 29 (cons 44 empty))))
  (cons (cons "McDavid" (cons 97 empty))
    (cons (cons "Draisatl" (cons 29 empty))
      (cons (cons "Kassian" (cons 44 empty)) empty))))
```

1.3. Odd ones in.

Write a function `odds-less-evens` which consumes a `(listof Int)` and produces an `Int` which is the result of adding up all the odd numbers in the list, but subtracting all the even numbers. For example,

```
(check-expect (odds-less-evens empty) 0)
(check-expect (odds-less-evens (cons 1 (cons 5 (cons 6 (cons 10 empty))))) -10)
```

1.4. Summing all lists.

Write a function `sum-all-lists` that consumes a non-empty `(listof (listof Num))` where all `(listof Num)` must have the same length. Your function should produce a single `(listof Num)` for which each entry $v_i$ is the result of summing the $i^{th}$ entry of each input `(listof Num)`. For example,

```
(check-expect (sum-all-lists (cons (cons 1 (cons 2 empty))
  (cons (cons 5 (cons 3 empty))
    (cons (cons 0 (cons 20 empty)) empty))))
  (cons 6 (cons 25 empty)))
```

Make sure to write the function `sum-lists` from 1.1 first, you should use it in your solution. Also think about how you might write this function using a `foldr`, what might your base be? How can you use that to help you write it here without a `foldr`?

---

2. Exit Strategies. When investing in the stock market it is important to have an exit strategy (holding to zero is not an ideal strategy). An investor can set up a stop-loss order and a limit-sell order to automatically list their stock for sale if the price gets too low, or when it reaches a high enough price to take profits respectively.

In a stop-loss order the investor sets a price limit that if the stock drops to or below that price they will immediately sell their shares at. For example, if the fictional stock GEEEMMY is currently priced at $20 per share and the investor sets a stop-loss order for $18, if GEEEMMY ever drops to $18 or below their shares will be immediately listed for sale at the current market price.

In a limit-sell order the investor sets a price limit that is the minimum price the investor will accept for their shares. If the the same fictional stock GEEEMMY is currently priced at $20 per share and the investor sets limit-sell order for $40, if the price ever raises to $40 or above the investors shares will also immediately be listed for sale at the current market price.
Write a function `exit-point` that consumes two `Num`s which represent the stop-loss price, and the limit-sell price respectively, as well as a non-empty non-negative `(listof Num)` which represents the stock price as it changes. Your function should produce the price that the investor sold at based on if their stop-loss or limit-sell triggered at any point. If neither order was triggered you should produce the last known price of the stock.

```racket
(check-expect (exit-point 10 50 (cons 20 (cons 30 (cons 8 (cons 90 empty)))))) 8
(check-expect (exit-point 5 300 (cons 20 (cons 45 (cons 88 (cons 10 (cons 15 empty)))))) 15)
(check-expect (exit-point 1 20 (cons 5 (cons 10 (cons 30 (cons 90 (cons 100 empty))))) 30)
```

3. Nutrition Info. Keeping track of nutrients with homemade recipes can be a pain. One has to sum up the nutrients of all the ingredients and then divide it by the number of servings made by the recipe. Since it’s such a pain we’d like to write a function to help us keep track.

We will use a `(listof Str)` to represent the names of all the nutrients we want to keep track of in our recipe. We will then use a `(listof (listof Num))` to represent the list of our ingredients, where each ingredient is represented by the `(listof Num)` that is value in grams it provides of each of the nutrients we’re tracking. That means each of our `(listof Num)` will have a number of elements equal to the number of elements in our `(listof Str)`. For example,

```racket
(define nutrients (cons "Fiber" (cons "Fats" (cons "Sugars" empty))))
(define ingredients (cons (cons 0 (cons 2 (cons 4 empty)))
                     (cons (cons 4 (cons 3 (cons 10 empty)))
                          (cons (cons 8 (cons 1 (cons 1 empty))) empty))))
```

the above shows a list of nutrients and a corresponding list of ingredients. The list of ingredients contains three ingredients - the first ingredient has 0 grams of Fiber, 2 grams of Fats, and 4 grams of Sugars.

Submit your solutions for this question in the file `nutrition.rkt`

Write a function `nutrition-per-serving` that consumes a `Nat` which represents how many servings the recipe makes, a `(listof Str)` which represents the nutrients we’re tracking, and a `(listof (listof Num))` which represents the ingredients and their nutritional information. Your function should produce a list of pairs where each pair begins with the nutrients name and the second value is the total grams of that nutrient per serving of the recipe.

For example,

```racket
(check-expect (nutrition-per-serving 3 nutrients ingredients)
              (cons (cons "Fiber" (cons 4 empty)) (cons (cons "Fats" (cons 2 empty))
                        (cons (cons "Sugars" (cons 5 empty)) empty))))
```

4. Rainbows. Wikipedia says that “a rainbow is a meteorological phenomenon that is caused by reflection, refraction and dispersion of light in water droplets resulting in a spectrum of light appearing in the sky.” You sometimes see them after rainfalls. They’re pretty.
The natural philosopher Isaac Newton labeled seven distinct colours in a rainbow, which are in order: 'red, 'orange, 'yellow, 'green, 'blue, 'indigo, and 'violet. But as we all know, rainbows often have some of the colours missing from them because unicorns steal the colours to make ice cream.

In Racket we can represent a valid rainbow as a list of colors with the following conditions:

- The list should contain only the colours: 'red, 'orange, 'yellow, 'green, 'blue, 'indigo, and 'violet.
- Colours must appear in that order.
- One or more of the colors may be missing due to unicorn theft.
- Colors can’t be repeated.

A rainbow without colours is still a rainbow (even though it is really just a damp patch of sky). Thus there are 127 possible rainbows that you could see in the sky, plus the empty rainbow, which you can’t see but is always there.

Solving some parts of this problem requires embellishment of the listof-X-template that you saw in class.

Submit your solutions for all parts of this question in the file rainbow.rkt.

### 4.1. A rainbow by any other name.

Write a predicate (rainbow? list) that determines if a list is a valid rainbow, according to the conditions above.

<table>
<thead>
<tr>
<th>(check-expect</th>
<th>(rainbow? (cons 'red (cons 'green (cons 'indigo empty))))) true</th>
</tr>
</thead>
<tbody>
<tr>
<td>(check-expect</td>
<td>(rainbow? (cons 'pineapple (cons 'pizza (cons pi empty))))) false</td>
</tr>
<tr>
<td>(check-expect</td>
<td>(rainbow? (cons 'red (cons 'red (cons 'blue empty))))) false</td>
</tr>
<tr>
<td>(check-expect</td>
<td>(rainbow? empty) true)</td>
</tr>
</tbody>
</table>
4.2. *Stolen rainbows.*

Write a function \((\text{unicorn colour rainbow})\) that produces a rainbow with the specified colour stolen from it. If the rainbow doesn’t contain the colour the rainbow is unchanged.

You can assume that the arguments are always a valid rainbow and colour according to the conditions above, i.e., these assumptions can form part of your contract.

\[
\begin{align*}
\text{check-expect} & \\
(\text{unicorn 'blue (cons 'red (cons 'yellow (cons 'blue empty))))} \\
(\text{cons 'red (cons 'yellow empty)})
\end{align*}
\]

\[
\begin{align*}
\text{check-expect} & \\
(\text{unicorn 'green (cons 'red (cons 'yellow (cons 'blue empty))))} \\
(\text{cons 'red (cons 'yellow (cons 'blue empty))})
\end{align*}
\]

4.3. *Leprechauns, of course it had to be leprechauns.* As you may have heard, leprechauns sometime steal the colours back from the unicorns and replace them in the rainbows.

Write a function \((\text{leprechaun colour rainbow})\) that produces a rainbow with the specified colour added to it. If the rainbow already contains the colour the rainbow is unchanged.

You can assume that the arguments are always a valid rainbow and colour according to the rules above, i.e., these assumptions can form part of your contract.

\[
\begin{align*}
\text{check-expect} & \\
(\text{leprechaun 'violet (cons 'red (cons 'yellow (cons 'blue empty))))} \\
(\text{cons 'red (cons 'yellow (cons 'blue (cons 'violet empty))))}
\end{align*}
\]

\[
\begin{align*}
\text{check-expect} & \\
(\text{leprechaun 'orange (cons 'red (cons 'yellow (cons 'blue empty))))} \\
(\text{cons 'red (cons 'orange (cons 'yellow (cons 'blue empty))))}
\end{align*}
\]

\[
\begin{align*}
\text{check-expect} & \\
(\text{leprechaun 'yellow (cons 'red (cons 'yellow (cons 'blue empty))))} \\
(\text{cons 'red (cons 'yellow (cons 'blue empty))})
\end{align*}
\]