CS 111 - Week 6 Lab Exercise

Deadline

Due by the end of lab. (Submit whatever you have by the end of lab, even if incomplete.)

How to submit

Submit your Definitions window (`lab6.rkt` file) and all of the `.txt` files that you end up creating for this lab using `~st10/111submit` on nrs-labs, with a homework number of **86**.

Purpose

To practice a little with additional types of `check-` expressions, and to practice using Racket file input/output.

Important notes

- You are required to work in **pairs** on this lab exercise. If you are not pair-programming, then you may not receive full credit for your lab exercise.
- If you have a question during lab, and I am helping another pair, add one or both of your names to the "Next:" list on the board, and I will get to you as soon as I can.
  - Asking other pairs for assistance while waiting is a good idea, as is helping other pairs around you!

Problem 1

Start your Racket Definitions window with comments including:

- both of your names,
- and that this is the Week 6 Lab Exercise.

Also include the usual `require` expressions, either before or after these comments.

You are **NOT** writing any functions for this problem - you are writing expressions to get some more experience using some of the `check-` functions besides `check-expect`.

1 part a

Write 3 expressions using `check-range` that result in passing tests if the expression

```
(+ 20 (random 11))
```

results in a value in [20, 30]. (You are giving the **same** expression **3** times, to see if three attempts given one after the other all work.)

Feel free to add additional `check-range` expressions if you would like.

Run these several times, until you are confident that these tests all should pass.

1 part b

Consider:

```
(define (play univ-num)
    (cond
        [(< (modulo univ-num 90) 30) "red"]
        [(< (modulo univ-num 90) 60) "green"]
        [else "blue"]
    )
)
```

Type this little function into your Definitions window.

Write 3 expressions using `check-member-of` with calls to `play`, each with an argument of `(random 500)`, that result in passing tests. (You again are giving the **same** expression **3** times, to see if three attempts given one after the other all work.)

Feel free to add additional `check-member-of` expressions if you would like.

Run these several times, until you are confident that these tests all should pass.

Problem 2

Include the BSL Racket file input/output functions using:

```
(require 2htdp/batch-io)
```

And, consider the file input/output examples given in lecture during Week 6 Lecture 2 (which are posted under "In-class Examples" on the CS 111 public course web site).

Again, you are **NOT** writing any functions for this problem - here, you are writing expressions to get some experience using some of file-related functions in `2htdp/batch-io`.

2 part a

Write an expression, using `write-file`, whose side-effects will be to create a file named `practice.txt` that contains at least 5 lines worth of contents of your choice, making sure that at least two lines contain **more** than one word each.

(Remember: `write-file` only expects two `string` arguments -- the name of the file to write to, and the string to write there -- but you can use `"\n"` within a string to say that you want a newline character.)

2 part b

Write an expression, using an appropriate `read-` function, whose value will be a single string containing all of the contents of `practice.txt`.

2 part c

Write an expression, using an appropriate `read-` function, whose value will be a **list** of strings, such that each string is one line from `practice.txt`.

2 part d

Write an expression, using an appropriate `read-` function, whose value will again be a **list** of strings -- but this time, each string will be one white-space-separated "word" from `practice.txt`.

2 part e

Write an expression that will write **both** of your names to a file `lab6-names.txt`.

Problem 3

3 part a

Consider: the function `length` produces the length of a list. And, consider that you know how to read a file's contents into a list of strings where each string is one line from a file.

Using the design recipe, design a function `get-num-lines` that expects the name of a file relative to the directory the Definitions window is saved in, and returns the number of lines in that file.

(Hint: this function's body should be quite short!)

3 part b

Also consider that you know how to read a file's contents into a list of strings where each string is one white-space-separated "word" in that file.

Using the design recipe, design a function `get-num-words` that expects the name of a file relative to the directory the Definitions window is saved in, and produces the number of white-space-separated "words" in that file.

(Hint: this function's body, also, should be quite short!)

Problem 4

Using the design recipe, write a function `form-letter` that expects a person's first name and the name of a file to write to, and returns the name of that given file, but **also** has the side effects of writing to that file a short customized message of your choice "to" that person, including their name at least 3 times in the resulting message.