Lab 6

Goals:

- More practice with list functions (lists of structures, lists of lists, etc.)
- Using Auxiliary functions
- Batch programs (file input)

Instructions:

Please submit all answers via Sakai in a single file named `lab6.rkt`. Do all work in pairs. Include the names of both partners at the top of the file in a comment. You need only submit one lab per pair.

This lab requires the `batch-io` library, `image` library, `states.rkt` , and `lab6-extras.rkt` and `states.json`

Download `lab6-extras.rkt`, `states.rkt` and `states.json` from the Sakai Resources area or lab attachments and save it in the same directory in which you are saving `lab6.rkt`. You will find the following lines at the beginning of the starter `lab6.rkt`:

```
(require 2htdp/batch-io)
(require 2htdp/image)
(require "lab6-extras.rkt")
(require "states.rkt")
```

It goes without saying, but you must use the Design Recipe for all functions. In particular,

- Don't forget to write a signature, purpose statement, and unit test(s) for every function.
- Write the test(s) before you write the function definition.
- ***Break up functions into smaller functions if they become too complex.***
- Define every type you use.
- Use the appropriate templates

Also: continue to work on Project 1. We will not require you to turn anything in for the project, but your goal for the week is to complete *Milestone 2*. You will turn something in for next week.

Problem 1: Representing Tweets

A Tweet consists of a username, the tweet message itself, a location (here, US state abbreviations), a date and a time. For example,

4/16/16	12:43	jacquijade	i have tonsillitis but at least i'm covered in cute kittens. @ The Little Clinic https://t.co/vGv8tyrrmo	CO

a. Develop a data definition corresponding to a Tweet. For this lab, you do not need to represent data and time as structures. If needed, you can use the data definition (used often in this lab)
```
;; StateAbrv is a two-letter String US postal code e.g. "DE", "PA", "DC", ...
```

b. Develop a data definition for a List of Tweets.

Problem 2: Working with Tweets

a. Design a function `tweets-that-contain` that consumes a List of Tweets and a String, and produces a List of Tweets from the given List of Tweets whose message contains the given string.

b. Design a function `tweets-from-state` that consumes a List of Tweets and a StateAbrv, and produces a List of Tweets from the given List of Tweets that were made in the given state.

Problem 3: Batch I/O

a. Develop a function `parse-tweet` that takes a RawTweet as its parameter and produces a Tweet. A RawTweet is defined as follows:

```
;; A RawTweet is (list String String String String StateAbbrev)
```

Assume that the elements in a RawTweet map to the fields of a Tweet as follows:

1. The first element is the date
2. The second element is the time
3. The third element is the Twitter username
4. The fourth element is the tweet message itself
5. The fifth element is the state abbreviation of the location (e.g., “DE”, “MA”)

To access the individual elements of a list, you may use the following functions: `first`, `second`, `third`, `fourth`, `fifth` — these are built-in.

Note that `parse-tweet` will *not* follow the template for a List-of-Strings function because a RawTweet is NOT a self-referential data definition.

b. Develop the Data Definition for a List of RawTweets. Remember to include a template and at least one example.

c. Develop a function `parse-tweets` that takes as its parameter a List of RawTweets and produces a List of Tweets.

d. Develop a function `load-tweets` that takes as its parameter a file name and produces a List of Tweets. The contents of files that are passed to load-tweets must be in CSV format. Each line in the file should be a comma separated list of strings that are formatted like the RawTweets for Problem 3a (i.e., the first element is the date, the second element is the time, etc.). **Your function should use the `read-csv-file` function provided by the batch-io library to read the contents of the given**

filename. `load-tweets` does NOT require unit tests (support for this is coming in the future, however!)

The file `tweets.csv` which is available from Sakai's Resources area or as an attachment to the lab contains approximately 57,000 actual tweets that were made on April 15-16, 2016. We also provide the file `tweets-short.csv` which has only the first 6000 lines (faster for testing). You can use these files as examples for this problem.

Note that, as these are actual tweets, they may contain profanity or other objectionable material. We do not endorse or condone any of the expressed viewpoints. If you believe you may be offended, please let one of the instructors know and we will provide a work around (although there is actually no reason to look at the individual tweets at all if you write your own tests!!)

Problem 4: Sentiment Analysis of Tweets

Sentiment analysis is a type of natural language processing that is concerned with identifying the attitude of a speaker or writer with respect to some topic or in general. It has a wide range of applications. For example, it can be used to gauge the public's reaction to a new product or to a presidential candidate's most recent speech.

In it's most basic form, sentiment analysis uses a mapping from words or phrases to sentiment scores to asses the overall sentiment score of a document. In this lab sentiment scores for a word will range from -5 to 5. Negative scores indicate a negative sentiment, positive scores indicate a positive sentiment (0 indicates no opinion), and the magnitude of the scores indicate the strength of the sentiment. You can examine AFINN-111.txt to see the sentiment scores for many English words. (Also check out AFINN-README.txt for some interesting information about how the sentiment score for the words were chosen.)

`lab6-extras.rkt` provides the `word->sentiment` function which takes as its parameter a string and produces the sentiment score for the given word.

```
; a SentimentScore is a NaturalNumber in [-5,5].

; word->sentiment: String -> SentimentScore
; Consumes:
;  String word: the word to score
; Produces: the SentimentScore for the given word
;
; (word->sentiment "sad") -> -2
; (word->sentiment "happy") -> 3
```

a. Copy the data definition of a List of Strings (including the template) from your lecture notes. Develop a function `avg-sentiment` that consumes a List of Strings and produces the average of the SentimentScores for all of the strings in the given List of Strings. In this case it is safe to assume the average sentiment of an empty tweet is 0, because of the semantics of "sentiment" (0 means neither positive nor negative).

b. Design a function `tweet-avg-sentiment` that takes a Tweet as its parameter and produces the average of the SentimentScores for the words in the message of the given Tweet. *The built-in `string-split` function can be used to split a given string into a list of the words in the given string.*

```
; string-split: String -> ListOfString
; Consumes:
;   String s: the string to split on white space
; Produces: a list of strings that contains the substrings in the
;               given string delimited by white space
;
; (string-split "hello world") -> (list "hello" "world")
```

c. Design a function `tweets-avg-sentiment` that takes as its parameter a List of Tweets and produces the average of the average SentimentScores of all of the Tweets in the given List of Tweets. Again, you can return 0 if the list is empty, but this case will need special handling in the final part (Problem 5)…

d. Design a function `state-sentiment-containing` that consumes a list of tweets, a state abbreviation, and a string, and computes the average sentiment of all tweets from that state that contain the given string. **USE FUNCTION COMPOSITION (this is a one-line function)**.

Problem 5: Putting it all together

The file "`states.rkt`" provides a constant US-STATE-ABRVS which is a list of the StateAbrv postal codes for all 50 states and "DC". It also provides a function `draw-state`:

```
;; draw-state :  StateAbrv Color Image --> Image

;; Consumes:
;;   StateAbbrev state:      a two-letter postal code
;;   Color       c:          the color to fill in the state with
;;   Image       background: a background image
;; Produces: an image of the given state solid filled with
;;           the given color, on the given background
;;           and at the proper relative location to every US state
```

You have already seen Color Strings like "blue" and "salmon", but you can also create a specific color by specifying a Red, Green, and Blue intensity (and optionally an Alpha channel transparency). Color intensity ranges in [0,255] with 0 being none and 255 max intensity of that color (0 is totally transparent and 255 is fully opaque for the alpha channel). For example,

```
;; make-color : Intensity Intensity Intensity [optional Intensity] -> Color
;; This function is part of htdp2e/images library

(make-color 255 0 0) ;Totally Red; default alpha is 255=opaque
(make-color 100 (random 255) (random 255)) ;100 Red, random green and blue
```

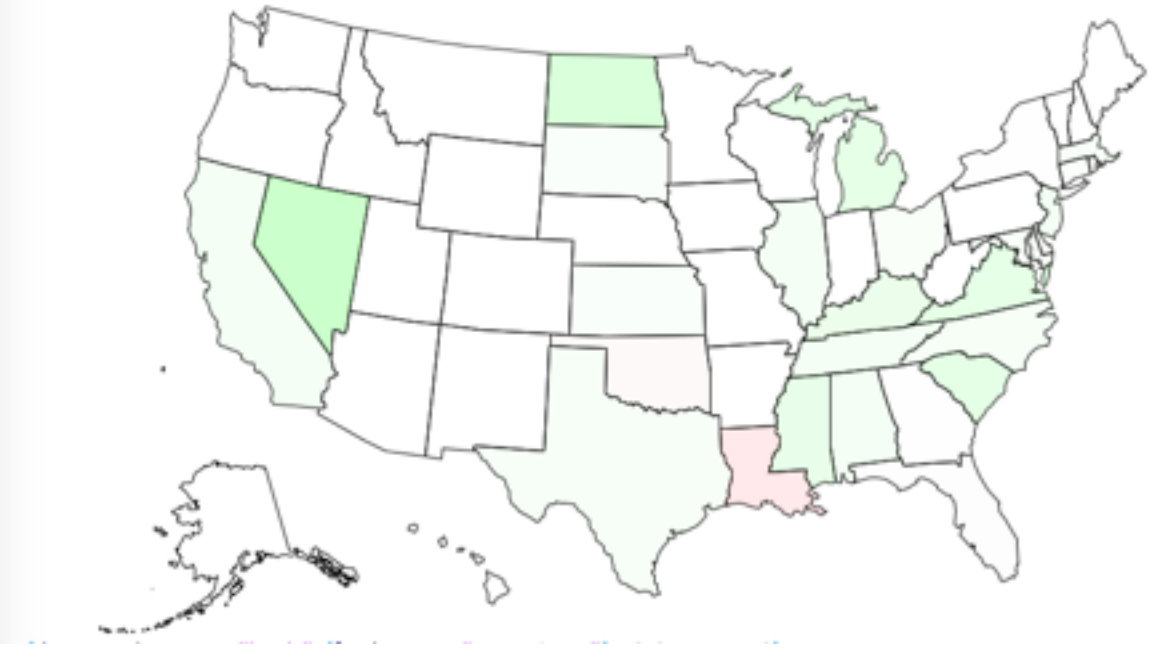
a. Design a function `draw-random-color-map` that draws the entire USA map with each state in a random color. *[This is a throw-away function for you to practice drawing a map; it can be used as a template for the next problem]*

b. Design a function `draw-sentiment-map` that consumes a string, and draws the map of the USA with each state colored according to its sentiment measured on tweets that contain the given string (**remember to use functions you wrote earlier**)

Recall that average sentiment will range from -5 to 5, so the function `sentiment-color`, provided by `states.rkt`, will produce a color from Red (-5) to Green (5) when given a sentiment value from -5 to 5.

```
;; a Sentiment is an Number in [-5,5]
;; sentiment-color : Sentiment --> Color
;; Consumes:
;;   Sentiment n: a sentiment value in [-5,5]
;; Produces: a color (red for negative, green for positive)
;;           and saturation (dark for strong,
;;           light for weak sentiments)
```

```
> (draw-sentiment-map "ford" (load-tweets "tweets3.csv") US-STATE-ABRVS)
```



Extra Credit

1. Real data is often noisy. For example, tweet messages often contain hash tags, @-mentions, emoticons, URLS, etc. that are not actually english words. To improve sentiment analysis, such noise is often filtered. Write a function `filter-message` that accepts as its parameter a List of Strings and produces a List of Strings with as much noise removed as possible. Make sure when you document this function, you explain not only what is being removed, but why you decided to remove it. After writing `filter-message`, modify your code appropriately and see if the result of the expression in Problem 5 changes. Did it change? Why do you think this is the case?
2. Create an interactive interface where you type a string, and the map displays the sentiment for that string using big-bang.
3. `parse-tweets` should really assume that the input file might be WRONG, that is, it should be checking for errors. Rewrite so that it detects errors in the state abbreviations, and also parses the dates and times in a reasonable way (to search for sentiment on a day or how it changes over time, etc.)