Lab 7 Regular

Goals: Practicing Abstraction & Polymorphism

Problem 1:

We have now added functions as a new atomic type, and the functions we write may consume (and produce--we haven't shown that yet...) functions. Up until now, all atomic types have come with an equality predicate [`=, string=?, image=?, boolean=?`]. What about functions?

1. Develop `some-function=?`. The function determines whether two functions from numbers to numbers produce the same results for 1.2, 3, and -5.775.
2. Can we hope to define `function=?`, which determines whether two functions from numbers to numbers are equal? ***Write your answer as a comment!***

Problem 2:

Abstract the following **two** functions into a **single** function named **`extreme`**:

```
;; mini : NELON  ->  number
;; to determine the smallest number
;; on alon
(define (mini alon)
  (cond
    [(empty? (rest alon)) (first alon)]
    [else (cond
            [(< (first alon)
                (mini (rest alon)))
             (first alon)]
            [else
             (mini (rest alon))])]))
```

```
;; maxi : NELON  ->  number
;; to determine the largest number
;; on alon
(define (maxi alon)
  (cond
    [(empty? (rest alon)) (first alon)]
    [else (cond
            [(> (first alon)
                (maxi (rest alon)))
             (first alon)]
            [else
             (maxi (rest alon))])]))
```

Both consume *Non-Empty Lists Of Numbers* ["NELON"] and produce a single number. The left one produces the smallest number in the list, the right one the largest.

Next, define `mini1` and `maxi1` in terms of the *abstracted* function `extreme`. Test each of them with the following three lists:

```
1.  (list 3 7 6 2 9 8)

2.  (list 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1)

3.  (list 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22)
```

Why are they slow on the long lists? ***Write your answer as a comment!*** (you can use the built-in function `time` to time a program, e.g. `(time (mini1 LIST1))` Remember to write a general, **parametric polymorphic type signature/contract** for your abstract function!!!

Problem 3. [refactoring problem 2]

Improve the *abstracted* function (.v2). Introduce a **`local`** name for *the result of the natural recursion*. Test `mini1.v2` and `maxi1.v2` with the same inputs again, and observe the timing.

Problem 4 [Finger exercises with abstract looping functions]

Abstract loops like `map`, `filter`, `foldr`, `sort`, `andmap`, and `ormap` are extremely useful and appear in many typical simple scripting/programming contexts. To practice we will again return to real data from the Internet, in this case Reddit.com posts. In particular, consider the following Data Definition:

```
;;(define-struct post
;;  (ups downs created
;;   subreddit id title author
;;   text? nsfw? content permalink))
;;make-post : Number Number Number
;;            String String String String
;;            Boolean Boolean String String) --> Post
;;INTERPRETATION
;;  ups [Number]: The number of upvotes associated with this Post.
;;  downs [Number]: The number of downvotes associated with this Post.
;;  created [Number]: The date that this Post was created.
;;  subreddit [String]: The subreddit that this Post was made in.
;;  id [String]: Unique ID; combination of letters, numbers, and dashes.
;;  title [String]: The title of this Post.
;;  author [String]: The username of the author of this Post.
;;  is-self [Boolean]: Is this Post pure text (True), or a URL (False).
;;  is-nsfw [Boolean]: Is this Post Not Safe for Work (NSFW).
;;  content [String]: The text of the post, or a url if not a text Post.
;;  permalink [String]: A permanent url that directs to this Post.
#| TEMPLATE
(define (post-fun ap)
  ... (post-ups ap) ;Number
  ... (post-downs ap) ;Number
  ... (post-created ap) ;Number
  ... (post-subreddit ap) ;String
  ... (post-id ap) ;String
  ... (post-title ap) ;String
  ... (post-author ap) ;String
  ... (post-text? ap) ;Boolean
  ... (post-nsfw? ap) ;Boolean
  ... (post-content ap) ;String
  ... (post-permalink ap) ;String
  )
|#
```

For testing purposes, we have two versions of a live, real-time Reddit feed. One uses cached data for a few subreddits so that you can write tests, and the other gives access to the actual real-time data feed. We will assume you are using the canned version, and will turn in that version, but once your code is working feel free to use the live version.

Attached to this lab are three files: `reddit-service-offline.rkt`, `reddit-service-online.rkt`, and `cache.json` (that contains the canned/cached data). Place these three files in the same directory as your `lab7.rkt` file, and place the line

```
(require "reddit-service-offline.rkt")
```

at the top (replace with the **`online`** version when you want to play with the live service).

The interface provided is the function `get-posts`:

```
;; a SortMode is an String enumeration
;;  -- "new"  [posts ranked by creation time]
;;  -- "hot"  [posts ranked by both votes and time]
;;
;; get-posts : String SortMode --> [list-of Post]
;; computes a real-time (or cached) list of posts for the given SubReddit
;; described in the first argument (e.g. "all", "movies", "news", etc.)
```

The OFFLINE cache provides the following SubReddit feeds: `"all"`, `"learnprogramming"`, `"crazyideas"`, `"cleanjokes"`, `"movies"`, `"news"`, and `"lifeprotips"`. *However, it may be easier to simply define 3 or 4 posts on your own with `make-post` for your unit tests.*

Solve the following problems with the built-in abstract looping functions. You may either use `local` or `lambda` to create the needed auxiliary function arguments.

1. Use `map` to define `posts->titles` that converts a list of Post values into a list of title Strings.
2. Use `map` to develop `posts->images` that converts a list of Post into a list of Images of the post's title string (use text) beside a triangle to the left of the text. The triangle should point up and be solid green if there are more up-votes than down-votes, otherwise red and downward-pointing. Create helpers as needed (using local and/or lambda)
3. Develop `sort-by-votes` that sorts a list of posts by (up-votes - down-votes). Use `sort`.
4. Use `filter` to develop `eliminate-NSFW` that removes any posts that are not safe for work.
5. Use `ormap` to develop `banned-author?` that consumes a string and a list of posts and returns true if any of the posts are by the given author.
6. Develop a function `select-posts` that consumes a list of Strings and a list of Posts, and produces a list of all the posts in the second list that contain any string in the first list. Hint: use `filter` and `string-contains?`. Consider helpers/auxiliaries if necessary.
7. Develop a function `too-wide` that consumes a length and a list of posts, and returns true if any post title is strictly longer than the provided string length. Should you use `andmap`, or `ormap`?? Defend your choice!!
8. Use `foldr` or `foldl` to develop a function to layout a list of images above one another, left-justified (hint: use `above/align` and `empty-image`). This should work with the result of question 4.2.

Problem 5: `Build-list`

`Build-list` is very useful for many purposes, especially to generate testing data. Let's practice. *All of the following should be defined with `build-list`.*

1. Develop a function to creates a list from 0 to n-1 for any natural number n
2. Develop a function to create a list from a to b inclusive for any natural numbers a and b.
3. Develop a function that creates a list of the first n even numbers.

Problem 6

Recall our definition of **`reduce`** in class. The built-in name in ISL for reduce is **`foldr`**:

```
;; foldr : (X Y  ->  Y) Y [listOf X]  ->  Y
;; (foldr f base (list x-1 ... x-n)) = (f x-1 ... (f x-n base))
(define (foldr f base alox) ...)
```

use **`foldr`** to define **`my-append`**, which does exactly what the builtin function **`append`** does, namely, juxtaposes the items of two lists or, equivalently, replaces **`empty`** at the end of the first list with the second list:

```
(check-expect (my-append (list 1 2 3) (list 4 5 6 7 8))
              (append    (list 1 2 3) (list 4 5 6 7 8)))
```

Problem 7.

We discussed in class that `fold/reduce` is VERY powerful. We defined `map` in terms of `fold/reduce`. Define **`my-filter`** (does what **`filter`** does) using **`foldr`**. [i.e. note that `foldr` can actually return lists, if you want...]