CS1114 Lists Problem Set

November 1, 2023

Instructions:

- Every student, however, must submit his or her own submission for credit.
- Questions do not necessarily appear in order of difficulty
- Each solution should be thoroughly **designed**, following each step of the design recipe in detail.

1. **Dynamic Survey Analysis**

   A company that conducts surveys regularly gets different results for their surveys, and changes their analysis over time. They want you to write a function, `perform-analysis` that takes in a list of responses and a function to analyze them. This way they can plug in different analysis functions for different occasions. For our purposes, each response will be a single numeric value. So as an example, given responses `'(5 3 4 5 2)` and an `average` function, your `perform-analysis` should return 3.8. If given a `median` function, it would return 4.

2. **Pricing Products**

   A warehouse wants a system where they can apply various discount strategies to their list of products. Design a function `apply-pricing-scheme` that takes a list of product prices and a discount function, returning the updated prices. For a first example, given prices `'(100 200 150)` and a 10% discount function, it should return `'(90 180 135)`. As a second example, if given those same prices `'(100 200 150)` and a function that computes 50% off all items under $150, and otherwise no discount, your function should return `'(50 200 150)`.

3. **Sort-criteria**

   A music app lets users sort their playlists based on specific criteria. While the app has a default way to sort songs (e.g., by title alphabetically), it allows power users to define their custom sorting preferences. Each song is represented as a record with the following data: length (in minutes and seconds), song title, artist name, and release year. Design a function `sort-criteria` that takes a list of songs and a user's function for comparing two songs. The user's comparison function will take two songs and return:

`#t` if the first song should come before the second song `#f` otherwise

For instance, if a user wants to sort songs based on their length (from shortest to longest), they would provide a comparison function that checks if the length of the first song is less than that of the second song. Your `sort-by` will function should use this comparison function to sort the given list of songs in the desired order.

4. **Alert Alert**

A software company is designing a customizable alert system for its users. The idea is that users will specify a threshold and the kind of alert they want to receive when a certain value exceeds (or falls below) that threshold.

Design a function called `create-alert` that:

- Takes in a threshold value and an alert type (either `"sound"` or `"vibration"`)
- Returns a new function. This new function should:
  - Take in a current value
  - If the current value exceeds the threshold and the alert type is `sound"`, it returns `Beep Beep!"`
  - If the current value exceeds the threshold and the alert type is `vibration"`, it returns `Vrrrrrr!"`
  - If the current value doesn't exceed the threshold, it returns `All quiet."`

For instance, suppose a user wants an alert when the temperature of their server exceeds 75 degrees, and they prefer a sound alert. They would use your `create-alert` function like this:

```
(define temperature-alert (create-alert 75 "sound"))
(temperature-alert 80) ; This should return "Beep Beep!"
(temperature-alert 70) ; This should return "All quiet."

(define stock-alert (create-alert 100 "vibration"))
(stock-alert 110) ; This should return "Vrrrrrr!"
(stock-alert 90) ; This should return "All quiet."
```

5. **Travel Planner**

You're creating software for a travel agency. They have two main functions: one that computes the total cost of a trip based on a list of hotel prices and another that computes the total duration based on a list of days spent in each hotel.

- One computes the total cost of a trip based on a list of hotel prices. If a hotel price is more than \$300, a discount of \$50 is applied to that particular hotel.
- Another that computes the total duration based on a list of days spent in each hotel. If a stay is longer than 5 days in a particular hotel, an extra day is added (e.g., for a weekend extension).

```
(define (total-cost hotels)
  (cond
    [(empty? hotels) 0]
    [(cons? hotels)
     (if (> (first hotels) 300)
         (+ (- (first hotels) 50) (total-cost (rest hotels)))
         (+ (first hotels) (total-cost (rest hotels))))]))

(define (total-days stays)
  (cond
    [(empty? stays) 0]
    [(cons? stays)
    (if (> (first stays) 5)
        (+ (+ 1 (first stays)) (total-days (rest stays)))
        (+ (first stays) (total-days (rest stays))))]))
```

Upon inspecting these two functions, you realize they have a lot of common structure. Both functions traverse the list and apply a certain condition on each element. Your task is to:

- Create an abstraction of these functions.
- Re-implement the `total-cost` and `total-days` functions using this abstraction.

6. **Inventory Management System**

A bookstore needs to process a list of orders made throughout the day. Each order is represented as a record containing the book's title, its price, and the quantity ordered. Each order can have either a positive or negative quantity ordered (i.e. returns). The bookstore wants to calculate the current running balance. Some book orders may require special processing to calculate the impact of that transaction (e.g. bulk discounts) so your program will take in a function for calculating the impact of an order on the balance.

Design a function called `process-orders` that takes:

- an initial account balance (a number).
- a list of orders (each order contains the book's title as a string, its price as a number, and the quantity ordered as a number).
- a function that consumes an order and an account balance, and produces the new account balance after including that order.
- Returns the final account balance after processing all the orders.

7. **Binary operations**

In the languages you have seen so far, we have worked with expressions like `(add1 6)`. Things like `add1`, `sub1` are specialized versions of `+` and `-`, specialized with `1` as the second argument. But we should want to create our own specialized versions. Write a function called `specialize-with`, that takes a binary operation on numbers and a number, and returns a function that has specialized that binary operation to a unary operation with a fixed second argument.

```
> (define mult7 (specialize-with * 7))
> (mult7 6)
42
```

8. **Specializing operations for values**

We saw in class that building different specialized operations can be tedious, and doing so for lots of operations could be more tedious still! Let's automate it. Write a function `make-all-unary-combinations` that takes a list of numbers and a list of binary functions over numbers, and then produces all options of unary functions specialized with that element as the second argument. So, for example, given `'(2 3)` and `(list - * +)`, it should produce a list of six unary (one-argument) functions that behave like "subtract 2 from", "subtract 3 from", "double", "triple", etc.

```
> (define lou (make-all-unary-combinations '(2 3) (list - * +)))
> ((first lou) 4)
2
> ((second lou) 4)
1
> ((third lou) 4)
8
```