CMPSC461 Fall - 2025
Programming Language Concepts

Dr. Suman Saha

Assignment-5 (Scheme Programming)

Instructions: For this assignment, you need to submit your solution as one single file named `code.rkt` to Gradescope. For any syntax-related help, refer to the official Racket docs (https://racket-lang.org). For all problems, you may assume all inputs obey the type constraints as specified in the problem. In other words, you do not need to perform any type checking as part of your solutions.

You are only allowed to use packages from the `#lang racket/base` library and cannot use any imperative features including assignment and loops. Use of non-imperative features from other packages, including those in the `racket/*` family, are not allowed. The autograder will reject submissions that include other libraries and will assign a zero until you resubmit.

Testing and Auto Grading: Gradescope's autograder will be used to grade this assignment. **You must follow the naming conventions for your submission and all functions as instructed, or the autograder will not be able to find your solutions**. To start, refer to the template `code.rkt` on Canvas, which has all of the expected functions already defined. Before the submission deadline, you can see your partial score based on a public set of tests on Gradescope. You can debug your code and resubmit as many times as is required before the deadline. After the deadline, the public tests *and other hidden tests* are used to calculate your final grade. The hidden tests account for no more than 40% of your total grade and are based directly on the requirements provided in the problem prompt.

Debugging Hints: If the autograder hangs for more than 5 minutes, or you see an error message like "The results uploaded to Gradescope were not formatted correctly", your submitted code has failed to terminate on at least one of the public or hidden private tests. The autograder cannot tell you which of these tests timed out, so carefully check your code to avoid non-terminating cases, infinite loops, and other errors that could prevent your functions from returning. Make sure you test extensively before submitting it on Gradescope, as we cannot regrade your submission after the assignment closes.

Note: *Gradescope integrates with a plagiarism checker that flags submissions that appear to be copied, shared, or otherwise duplicated. Submission suspected of plagiarism will be referred to the instructor. See the course syllabus for restrictions related to academic integrity.*

Problem 1: Working with Data [$2 + 3 + 6 + 4 = 15$pts]

Part 1.1 (2 pts): Complete the `clamp-bounds` function, which takes in an upper bound, a lower bound, and a list of positive and negative integers. This function returns a new list where all entries above and below their respective bounds are replaced by those bounds. This new list should be the same length as the input list. Assume that the lower bound will always be less than or equal to the upper bound. Your function must work for all sizes of lists.

Input			Output
`9`	`-9`	`'()`	`'()`
`-3`	`-7`	`'(-4 -5)`	`'(-4 -5)`
`1`	`1`	`'(1 2 3)`	`'(1 1 1)`
`2`	`-1`	`'(2 -4 3)`	`'(2 -1 2)`

Part 1.2 (3 pts): Complete the function `cleanup-data`, which takes in a list and does the following for each item:

- If it's a boolean, remove it.
- If it's a string, capitalize it.
- If it's a number, divide by 2 if it's even and multiply by 3 and subtract 1 if its odd.
- If it's a list, recurse over that list and append the result.

The input list will only contain an arbitrary nesting of booleans, strings, and numbers. The output should be a list that contains nothing but strings and numbers.

Hint: Racket has many functions that could help simplify handling strings. Refer to the Racket documentation for more information.

Input	Output
`'(((())))`	`'()`
`'(#t "2" 3)`	`'("2" 8)`
`'(1 (2 (3 (4))))`	`'(2 1 8 2)`
`'(2 "we" (#f 1 "are" #t) 6)`	`'(1 "WE" 2 "ARE" 3)`

(*continues on next page*)

Problem 1: Working with Data (continued)

Part 1.3 (6 pts): Complete the function lesser-decadents, which takes a list of numbers and reorganizes them such that each item in the list becomes a pair of values. The first item in this pair should be the original value, while the second is the list of items that occur after and are strictly less than the current value. Consider the input `'(1 3 4 2 3)` which is shown below. The first 3 is paired with a list that contains 2 since that is the only value after that 3 that is strictly less. The 4 is paired with `'(2 3)` as both the 2 and 3 occur after and are strictly less.

Input	Output
`'()`	`'()`
`'(1 2 3)`	`'((1 ()) (2 ()) (3 ()))`
`'(-2 5 0)`	`'((-2 ()) (5 (0)) (0 ()))`
`'(1 3 4 2 3)`	`'((1 ()) (3 (2)) (4 (2 3)) (2 ()) (3 ()))`

Part 1.4 (4 pts): The windowed average (also called a *simple moving average*) is the unweighted mean of all possible consecutive subsets of size N. Starting at the first index, we average the next N entries before increasing our index by 1. This process repeats until there are fewer than N items remaining in the list. Consider the inputs `4` and `'(1 2 3 4 5 6)`. There are 3 windows of this length in the list which results in the list `'(2.5 3.5 4.5)`.

1	2	3	4	5	6	Input Data, $N = 4$	
1	2	3	4			1st Window	$(1 + 2 + 3 + 4)/4 = 2.5$
	2	3	4	5		2nd Window	$(2 + 3 + 4 + 5)/4 = 3.5$
		3	4	5	6	3rd Window	$(3 + 4 + 5 + 6)/4 = 4.5$

When $N = 1$, we are essentially copying the list. When $N >$ length, there no valid windows, so we return an empty list. When $N < 1$, the problem is undefined and we also return an empty list.

Complete the function `windowed-average`, which implements this algorithm. This function should take two arguments, the size and a list of integers, and return a list. The input list unordered, includes both positive and negative integers, and can be of any size.

Note: Racket will delay evaluation of pure numeric division to preserve accuracy, so you might see `'(2/5 2/7 2/9)` instead of decimals. These are fractions, *not* division operations. Remember that in Scheme division is written as `(/ 2 5)`, not `(2 / 5)`. Refer to the Racket documentation on *exactness* for more information.

Problem 2: Dealing with Durations [$4 + 8 + 8 = 20$pts]

You are in charge of writing a program uses lists of one to three positive integers representing a duration of hours, minutes, and/or seconds. A duration might not be any longer than a few seconds, so your code must accept these values in the form `'(seconds)`, `'(minutes seconds)`, or `'(hours minutes seconds)`. Note that the same duration can be represented in multiple ways. For example, `'(10 0)` and `'(0 10 0)` both represent 10 minutes.

Part 2.1 (4 **pts):** Implement `pad-duration`, which takes in a single duration and inserts zeros for any missing fields. Order of values should remain the same so that the input and output are equivalent. For example, the input `'(10 20)` should result in `'(0 10 20)'`, which both represent 10 minutes and 20, not `'(20 10 0)` which would be 20 hours and 10 minutes.

Input	Output
`'(123)`	`'(0 0 123)`
`'(45 67)`	`'(0 45 67)`
`'(1 2 3)`	`'(1 2 3)`
`'(0 0 0)`	`'(0 0 0)`

Part 2.2 (8 **pts):** Implement `fmt-duration`, which takes in a single duration and converts it into a human-readable string. The hour, minute, and second values should be plural when appropriate and be separated by a comma and the phrase "and" while omitting any values that are zero. If all values are zero, you should output `"0 Seconds"`. **Be careful to match the specified format!**

Hint: `number->string` converts a number into a string and `string-append` appends strings together. Refer to the Racket documentation for more information.

Input	Output
`'(99)`	`"99 Seconds"`
`'(1 0 0)`	`"1 Hour"`
`'(9 1)`	`"9 Minutes and 1 Second"`
`'(1 2 3)`	`"1 Hour, 2 Minutes, and 3 Seconds"`

Part 2.3 (8 **pts):** Write the function 'add-durations', which takes in two durations and adds them together. The resulting duration should be in the simplest possible form. This means that when the count of seconds is greater than 59, it overflows into minutes. Similarly, when the count of minutes is greater than 59, it overflows into hours. Leading zeros in the hours or minutes places should also be removed so the resulting duration is as small as possible.

Hint: While calculating, it might be useful to use `pad-duration` to normalize the inputs.

Inputs		Output
`'(0 0 0)`	`'(10)`	`'(10)`
`'(123)`	`'(1 0 0)`	`'(1 2 3)`
`'(45 67)`	`'(8 9)`	`'(54 16)`
`'(0 30 0)`	`'(30 0)`	`'(1 0 0)`

Problem 3: Curried Calculations [15pts]

Restaurants here in State College get busy during football weekends, with thousands of people traveling into the city to take part in the festivities. A local Indian restaurant has asked you to help rebuild its order-tracking system to keep up with the high volume of orders. Implement `calculate-subtotal`, which takes a list representing the order to process and a list of menu items discounted, and returns the subtotal (or pre-tax) cost of the order, rounded to 2 decimal places.

The first parameter is a list of integers and strings representing the items a customer ordered. Each integer represents how many of the following items were ordered. Each string is the name of an item on the menu. You are guaranteed that each integer will be at least 1 and be followed by a string, and that each string is a valid item from the menu. For example, an order for two `"coffee"` and a `"mango lassi"` could look like any of the following examples. Note that it doesn't matter if we omit the 1 or the order in which the items occur.

```
'(2  "coffee"          "mango lassi")
'(2  "coffee"       1  "mango lassi")
'(   "mango lassi"  2       "coffee")
'(1  "mango lassi"  2       "coffee")
```

The second parameter is a list of name/integer pairs representing any items currently on sale, along with their discount rates. The rate is an integer between 1 and 99. While the name is always a valid menu item, the discounts list may contain only a subset of the menu's items or be empty. If `"mango lassi"` is 15% off and `"fish curry"` is 20% off, the discounts list could look like `'(("mango lassi" 15) ("fish curry" 20))`. As an example of everything working together, an order for `"aloo tikki"`, two `"chicken saag"`, one `"malai kofta"`, and two `"spiced tea"` has a subtotal of $55.26 when the `"malai kofta"` has a 10% discount.

While helping design this software, expect the following menu items:

Appetizers		Main Course		Drinks	
`"vegetable samosa"`	$4.99	`"fish curry"`	$16.99	`"mango lassi"`	$5.00
`"aloo tikki"`	$5.35	`"tikka masala"`	$16.35	`"spiced tea"`	$3.50
`"paneer pakora"`	$6.50	`"chicken saag"`	$15.00	`"coffee"`	$3.50
		`"daal tadka"`	$13.10		
		`"malai kofta"`	$14.35		

Hint: Rounding can be done using the following procedure: multiply your number so all the decimal points you care about are whole numbers, use the `round` function which will truncate the number to an integer, and then divide so the decimals are restored.