## Plan-Composition Study Problems

### Rainfall

Design a program called *rainfall* that consumes a list of real numbers representing daily rainfall readings. The list may contain the number -999 indicating the end of the data of interest. Produce the average of the non-negative values in the list up to the first -999 (if it shows up). There may be negative numbers other than -999 in the list (representing faulty readings).

Example: rainfall([1, -2, 5, -999, 8]) yields 3

### Palindrome Detection Modulo Spaces and Capitalization

A palindrome is a string with the same letters in each of forward and reverse order (ignoring capitalization). Design a program called *isPalindrome* that consumes a string and produces a boolean indicating whether the string with all spaces and punctuation removed is a palindrome.

Treat any character other than a letter or digit as punctuation.

Example: isPalindrome(“a man, a plan, a canal: Panama”) yields true.

isPalinidrome(“abca”) yields false

### Shopping Discount

An online clothing store applies discounts during checkout. A shopping cart is a list of the items being purchased. Each item has a description (a string like “shoes”) and a price (a real number like 12.50). Design a program called *checkout* that consumes a shopping cart and produces the total cost of the cart after applying the following two discounts:

* if the cart contains at least 100 worth of shoes, take 20% off the cost of all shoes (match only items whose exact description is “shoes”)
* If the cart contains at least two hats, take 10 off the total of the cart (match only items whose exact description is “hat”)

Example: checkout([(“shoes”, 25), (“bag”, 50), (“shoes”, 85), (“hat”, 15)]) yields 153 (175 total for the cart minus 20% of 110 worth of shoes)

### Sum of Squares

Design a program *sumOfSquares* that consumes an integer *n* and produces the sum of the squares of all numbers from 1 through *n*. You may assume that *n* is at least 1.

Example: sumOfSquares(4) yields 30

### Sum Over Table

Assume that we represent tables of numbers as lists of rows, where each row is itself a list of numbers. The rows may have different lengths. Design a program *sumLargest* that consumes a table of numbers and produces the sum of the largest item from each row.

Example: to run this program on the table shown below, use sumLargest([[1,7,5,3],[20],[6,9]]), which yields 36 (7 + 20 + 9).

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 7 | 5 | 3 |
|  | 20 |  |  |
| 6 |  |  | 9 |

### Length of Triples

Design a program called *maxTripleLength* that consumes a list of strings and produces the length of the longest concatenation of three consecutive elements. Assume the input contains at least three strings.

Example: maxTripleLength([“a”, “bb”, “c”, “dd”]) yields 5 (from “bb”, “c”, and “dd”)