**Problem 01: 10 points**

Topic: String and List Comprehension, Functions

**Part a: 4 points**

Write a function, **appendAll**, that takes two lists of strings as input and appends each element of the first list to each element of the second list. **appendAll** returns a list of the appended strings. Below is some sample test cases and their corresponding outputs:

\*Main> appendAll ["hello " , "good ", "bad "] ["there", "day!"]

["hello there","hello day!","good there","good day!","bad

there","bad day!"]

\*Main> appendAll ["hello " , "good ", "bad "] [" morning",

"day!"]

["hello morning","hello day!","good morning","good day!","bad

morning","bad day!"]

\*Main> appendAll ["sunny ", "good "] ["day!"]

["sunny day!","good day!"]

**- Hint:**

- You may find append function (`++`) useful to solve this

problem.

- You are not allowed to write a solution that uses the

recursive approach or higher-order functions.

**Part b: 3 points**

Write a function, **allThatContainsY**, that takes a list of strings and a character as input and returns a list of those strings that have the character in them.

|  |
| --- |
| \*Main> allThatContainsY ["goal", "Guide", "giggle", "brew"] 'g'  ["goal","giggle"]    \*Main> allThatContainsY ["goal", "Guide", "giggle", "brew"] 'G'  ["Guide"]    \*Main> allThatContainsY ["goal", "Guide", "giggle", "brew"] 'w'  ["brew"]    \*Main> allThatContainsY ["goal", "Guide", "giggle", "brew"] 'e'  ["Guide","giggle","brew"] |
|  |

**Part c: 3 points**

Write a function, **appendAndReturnOnlyWithY**, that takes two lists of strings and a character as input, appends every element of the first list to every element of the second list, and returns a list of those appended strings only if they contain y.

\*Main> appendAndReturnOnlyWithY ["sweet", "nice"] [" dreams"] 's' ["sweet dreams","nice dreams"]

\*Main> appendAndReturnOnlyWithY ["sweet", "nice"] [" dreams"] 'w' ["sweet dreams"]

\*Main> appendAndReturnOnlyWithY ["sweet", "bad", "quit"] [" dreams", " home", " it"] 't'

["sweet dreams","sweet home","sweet it","bad it","quit dreams","quit home","quit it"]

- **Hint:**

- You may use functions written for part *a* and *b* in order to

solve part *c*.

- You may find Chapter 1, 2, and 3 of your textbook useful

to solve these problems.

- Do not worry about specifying the function types.

**Problem 02: 6 points**

Topic: Types and Typeclasses

Briefly describe the types of **funny, expr**, and **t**. In other words, interpret the output of **:t funny**, **:t expr**, and **:t t** returned by **GHCi**.

-- a

\*Main> funny xx yy zz = (xx ++ zz) == yy

\*Main> :t funny

funny :: Eq a => [a] -> [a] -> [a] -> Bool

--b

\*Main> expr x = [ z | z <- [1.. (sum [1..x])]]

\*Main> :t expr

expr :: (Num a, Enum a) => a -> [a]

-- c

\*Main> t = (3.5 :: Float) : [0.1, 0.2, 0.3]

\*Main> :t t

t :: [Float]

Hint:

-You may find the following site useful:

o https://hackage.haskell.org/package/base- 4.16.0.0/docs/GHC-Enum.html

o https://hackage.haskell.org/package/base- 4.16.0.0/docs/GHC-Num.html

**Problem 03: 9 points**

Topic: Functions and Conditionals

a) Write a function f(x) such that -

*f(x) = 0.5 when x > 0*

*f(x) = -0.5 when x < 0*

*f(x) = 0.0 when x = 0*

b) Write another function 𝑔(𝑥) = 𝑥^2 + 2𝑥 + 𝑓(𝑥).

c) Write a function 𝑘(𝑛, 𝑥) so that it generates a list of numbers

bycalling 𝑔(𝑥 + 𝑖) where −𝑛 <= 𝑖 <= 𝑛.

𝑘(𝑛, 𝑥) = ∀!"#$#" 𝑔(𝑥 + 𝑖)

- Hint:

- In every case, 𝑥 is a number. - 𝑛 is a positive integer.

The following block may help you understand the problem.

-- testing function f

\*Main> f 1

0.5

\*Main> f (-1)

-0.5

\*Main> f (-10)

-0.5

\*Main> f 10.5

0.5

\*Main> f 10.555

0.5

\*Main> f (10/555)

0.5

-- testing function g

\*Main> g 1

3.5

\*Main> g (-1)

-1.5

\*Main> g (0)

0.0

\*Main> g (2)

8.5

\*Main> g (-1.5)

-1.25

\*Main> g (-2)

-0.5

-- testing k

\*Main> k 0 1

[3.5]

\*Main> k 0 0

[0.0]

\*Main> k 0 2

[8.5]

\*Main> k 1 3

[8.5,15.5,24.5]

\*Main> k 1 1

[0.0,3.5,8.5]

\*Main> k 2 1

[-1.5,0.0,3.5,8.5,15.5]

\*Main> k 3 1

[-0.5,-1.5,0.0,3.5,8.5,15.5,24.5]

**Problem 04: 05 points**

Topic: What does it do?

What does the function, **goodBad,** do? Explain in your own words. Add three test cases.

goodBad :: (Num a, Enum a) => [a] -> [a]

goodBad xs = [product [1..z]| z <- reverse([2 \* x | x<-xs])]