Haskell Programming Assignment: Various Computations

Abstract: Programming exercises that focus on functions, recursive list processing, list comprehensions, and higher order functions in Haskell.

Mindful Mimicking:

Demo

```
GHCi, version 9.2.2: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> length [2,3,5,7]
4
>>> words "need more coffee"
["need","more","coffee"]
>>> unwords ["need", "more", "coffee"]
"need more coffee"
>>> reverse "need more coffee"
"eeffoc erom deen"
>>> reverse ["need", "more", "coffee"]
["coffee","more","need"]
>>> head ["need", "more", "coffee"]
"need"
>>> tail ["need", "more", "coffee"]
["more","coffee"]
>>> last ["need", "more", "coffee"]
"coffee"
>>> init ["need", "more", "coffee"]
["need","more"]
>>> take 7 "need more coffee"
"need mo"
>>> drop 7 "need more coffee"
"re coffee"
>>> (\x -> length x > 5) "Friday"
True
>>> (\x -> length x > 5) "uhoh"
False
>>> (\x -> x /= ' ') 'Q'
True
>>> (\x -> x /= ' ') ' '
False
>>> filter (\x -> x /= ' ') "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>> :quit
Leaving GHCi.
```

Numeric Function Definitions:

Demo

```
GHCi, version 9.2.2: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :l task2.hs
[1 of 1] Compiling Main             ( task2.hs, interpreted )
Ok, one module loaded.
>>> squareArea 10
100.0
>>> squareArea 12
144.0
>>> circleArea 10
314.15927
>>> circleArea 12
452.38934
>>> blueAreaOfCube 10
482.19025
>>> blueAreaOfCube 12
694.354
>>> blueAreaOfCube 1
4.8219028
>>> map blueAreaOfCube [1..3]
[4.8219028,19.287611,43.397125]
>>> paintedCube1 1
0
>>> paintedCube1 2
0
>>> paintedCube1 3
6
>>> paintedCube2 1
0
>>> paintedCube2 2
0
>>> paintedCube2 3
12
>>> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
>>> :quit
```

Numeric Function Definitions:

Code

```haskell
squareArea :: Float -> Float
squareArea sideLength = sideLength * sideLength

circleArea :: Float -> Float
circleArea radius = pi * (radius * radius)

blueAreaOfCube :: Float -> Float
blueAreaOfCube sideLength = 6 * (blueArea - whiteArea)
  where
    r = sideLength / 4
    blueArea = squareArea sideLength
    whiteArea = circleArea r

paintedCube1 :: Int -> Int
paintedCube1 degree
  | degree <= 2 = 0
  | otherwise = 6 * (degree - 2) ^ 2

paintedCube2 :: Int -> Int
paintedCube2 degree
  | degree <= 2 = 0
  | otherwise = 12 * (degree - 2)
```

Puzzlers:

Demo

```
GHCi, version 9.2.2: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load task3.hs
[1 of 1] Compiling Main             ( task3.hs, interpreted )
Ok, one module loaded.
>>> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
>>> reverseWords "want me some coffee"
"coffee some me want"
>>> averageWordLength "want me some coffee"
4.0
>>> averageWordLength  "appa and baby yoda are the best"
3.5714285
>>> :quit
Leaving GHCi.
```

Puzzlers:

Code

```haskell
-- function composition!
reverseWords :: String -> String
reverseWords = unwords . reverse . words

averageWordLength :: String -> Float
averageWordLength s = fromIntegral totalSentenceLength / fromIntegral (length sentWords)
  where
    wordLengths = map length (words s)
    totalSentenceLength = sum wordLengths
    sentWords = words s
```

Recursive List Processors:

Demo

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
>>> :l task4.hs
[1 of 1] Compiling Main             ( task4.hs, interpreted )
Ok, one module loaded.
>>> list2Set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
>>> list2Set "need more coffee"
"ndmr cofe"
>>> isPalindrome ["coffee", "latte", "coffee"]
True
>>> isPalindrome ["coffee", "latte", "espresso", "coffee"]
False
>>> isPalindrome [1,2,5,7,11,13,11,7,5,3,2]
False
>>> isPalindrome [2,3,5,7,11,13,11,7,5,3,2]
True
>>> collatz 10
[10,5,16,8,4,2,1]
>>> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>> collatz 100
[100,50,25,76,38,19,58,29,88,44,22,11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
>>> :quit
Leaving GHCi.
```

Recursive List Processors:

Code

```haskell
list2Set :: Eq a => [a] -> [a]
list2Set [] = []
list2Set (x : xs)
  | x `elem` xs = list2Set xs
  | otherwise = x : list2Set xs

isPalindrome :: (Eq a, Show a) => [a] -> Bool
isPalindrome [] = True
isPalindrome [x] = True -- (x:_) is the same as [x]!
isPalindrome (x : xs) = x == last xs && isPalindrome (init xs)
```

```haskell
collatz :: Int -> [Int]
collatz n
  | n == 1 = [1]
  | even n = n : collatz (n `div` 2)
  | odd n = n : collatz (3 * n + 1)
  | otherwise = []
```

Higher Order Functions:

Demo

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
>>> :l task5.hs
[1 of 1] Compiling Main             ( task5.hs, interpreted )
Ok, one module loaded.
>>> count 'e' "need more coffee"
5
>>> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
>>> freqTable "need more coffee"
[('n',1),('d',1),('m',1),('r',1),(' ',2),('c',1),('o',2),('f',2),('e',5)]
>>> freqTable [1,2,3,2,3,4,3,4,5,4,5,6]
[(1,1),(2,2),(3,3),(4,3),(5,2),(6,1)]
>>> :quit
Leaving GHCi.
```

Higher Order Functions:

Code

```haskell
-- list2Set from task4.hs
list2Set :: Eq a => [a] -> [a]
list2Set [] = []
list2Set (x : xs)
  | x `elem` xs = list2Set xs
  | otherwise = x : list2Set xs

count :: Eq a => a -> [a] -> Int
count x xs = length [s | s <- xs, s == x]

freqTable :: Eq a => [a] -> [(a, Int)]
freqTable items = [(i, count i items) | i <- setOfItems]
  where
    setOfItems = list2Set items
```

An Interesting Statistic: nPVI

Demo

Data Test:

a,b,c,d,e,f,g

```haskell
-- nVPI implementation in Haskell

-- Test data
a :: [Int]
a = [2, 5, 1, 3]

b :: [Int]
b = [1, 3, 6, 2, 5]


c :: [Int]
c = [4, 4, 2, 1, 1, 2, 2, 4, 4, 8]

u :: [Int]
u = [2, 2, 2, 2, 2, 2, 2, 2, 2, 2]

x :: [Int]
x = [1, 9, 2, 8, 3, 7, 2, 8, 1, 9]

pairwiseValues :: [Int] -> [(Int, Int)]
pairwiseValues xs = zip (init xs) (tail xs)

pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences xs = map (\(x, y) -> x - y) $ pairwiseValues xs

pairwiseSums :: [Int] -> [Int]
pairwiseSums xs = map (\(x, y) -> x + y) $ pairwiseValues xs

half :: Int -> Double
half n = fromIntegral n / 2

pairwiseHalves :: [Int] -> [Double]
pairwiseHalves = map (\x -> fromIntegral x / 2)

pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums = pairwiseHalves . pairwiseSums

pairwiseTermPairs :: [Int] -> [(Int, Double)]
pairwiseTermPairs xs = zip (pairwiseDifferences xs) (pairwiseHalfSums xs)

term :: (Int, Double) -> Double
term nd = abs (fromIntegral (fst nd) / snd nd)

pairwiseTerms :: [Int] -> [Double]
pairwiseTerms xs = map term (pairwiseTermPairs xs)

nPVI :: [Int] -> Double
nPVI xs = normalizer xs * sum (pairwiseTerms xs)
  where
    normalizer xs = 100 / fromIntegral ((length xs) - 1)
```

Historic Code: Dit Dah Code

Demo

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
*Main> dit
"-"
*Main> dah
"---"
*Main> dit +++ dah
"- ---"
*Main> m
('m',"--- ---")
*Main> g
('g',"--- --- -")
*Main> h
('h',"- - - -")
*Main> symbols
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -"),('d',"--- - -"),('e',"-"),('f',"- - --- -"),('g',"--- ---
 -"),('h',"- - - -"),('i',"- -"),('j',"- --- --- ---"),('k',"--- - ---"),('l',"- --- - -"),('m',"--- ---"),('n
',"--- -"),('o',"--- --- ---"),('p',"- --- --- -"),('q',"--- --- - ---"),('r',"- --- -"),('s',"- - -"),('t',"-
"),('u',"- - ---"),('v',"- - - ---"),('w',"- --- ---"),('x',"--- - - ---"),('y',"--- - --- ---"),('z',"--- -
-- - -")]
*Main>
```

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
*Main> assoc 'a' symbols
('a',"- ---")
*Main> assoc 'z' symbols
('z',"--- --- - -")
*Main> find 'b'
"--- - - -"
*Main> find 'y'
"--- - --- ---"
*Main>
```

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
*Main> addletter (encodeletter 'a') (encodeletter 'b')
"- ---   --- - - -"
*Main> addword (encodeword "hello") (encodeword "haskell")
"- - - -   -   - --- - -   - --- - -   --- --- ---       - - - -
-   - - -   --- - ---   -   - --- - -   - --- - -"
*Main> drop
drop       dropWhile  droplast3  droplast7
*Main> droplast3 [1,2,3,4,5,6]
[1,2,3]
*Main> droplast7 [1,2,3,4,5,6,7,8,9,10]
[1,2,3]
```

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
*Main> encodeletter 'm'
"--- ---"
*Main> encodeletter 'a'
"- ---"
*Main> encodeletter 'd'
"--- - -"
*Main> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
*Main> encodeword "mad"
"--- ---   - ---   --- - -"
*Main> encodeword "haskell"
"- - - -   - ---   - - -   --- - ---   -   - --- - -   - --- - -"
*Main> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -
  --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
*Main> encodemessage "i ran out of ram"
"- -       - --- -   - ---   --- -       --- --- ---   - - ---   ---
    --- --- ---   - - --- -       - --- -   - ---   --- ---"
*Main> encodemessage "time to download some more"
"---   - -   --- ---   -       ---   --- --- ---       --- - -   --- --
- ---   - --- ---   --- -   - --- - -   --- --- ---   - ---   --- - -
     - - -   --- --- ---   --- ---   -       --- ---   --- --- ---   -
--- -   -"
```