Haskell Programming Assignment: Various Computations

Learning abstract

This Assignment features programs that focus on functions, recursive list processing, list comprehensions, and higher order functions in Haskell.

Task 1 - Mindfully Mimicking the Demo

```
C:\Users\trist> ghci
GHCi, version 9.2.5: 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.
```

Task 2 - Numeric Function Definitions

Code:

ass1.hs

```haskell
squareArea :: Num s => s -> s
squareArea side = side * side

circleArea :: Fractional r => r -> r
circleArea radius = 3.141592653589793 * (radius * radius)

blueAreaOfCube :: Fractional s => s -> s
blueAreaOfCube side = sideArea * 6
    where sideArea = squareArea side - circleArea radius
          radius = side * (1 / 4)

paintedCube1 :: (Ord s, Num s) => s -> s
paintedCube1 n =
    if (n < 3) then 0 else
        numberOfSides * 6
        where numberOfSides = (n - 2) * (n - 2)

paintedCube2 :: (Ord s, Num s) => s -> s
paintedCube2 n =
    if (n < 3) then 0 else
        numberOfSides * 6
        where numberOfSides = (n - 2) * 2

```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l ass1.hs
[1 of 1] Compiling Main             ( ass1.hs, interpreted )
Ok, one module loaded.
ghci> :set prompt ">>>"
>>> squareArea 10
100
>>>squareArea 12
144
>>> circleArea 10
314.1592653589793
>>> circleArea 12
452.3893421169302
>>> blueAreaOfCube 10
482.19027549038276
>>> blueAreaOfCube 12
694.3539967061512
>>> blueAreaOfCube 1
4.821902754903828
>>> map blueAreaOfCube [1...3]

<interactive>:10:23: error:
    Variable not in scope: (...) :: t0 -> t1 -> b
>>>map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
>>> paintedCube1 1
0
>>>paintedCube1 2
0
>>> paintedCube1 3
6
>>> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
>>> paintedCube2 1
0
>>>paintedCube2 2
0
>>> paintedCube2 3
12
>>> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
>>> :quit
Leaving GHCi.
```

Task 3 - Puzzlers

Code:

puzzlers.hs

```haskell
reverseWords inputWords = unwords(reverse(words inputWords))

averageWordLength inputWords =
    fromIntegral (wordLength - spaces) / fromIntegral (wordCount)
        where wordLength = length inputWords
              wordCount = length (words inputWords)
              spaces = wordCount - 1

```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l "puzzlers.hs"
[1 of 1] Compiling Main             ( puzzlers.hs, interpreted )
Ok, one module loaded.
ghci> :set prompt ">>>"
>>> 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 "appa and baby yoda are the best"
3.5714285714285716
>>>averageWordLength "want me some coffee"
4.0
>>>:quit
Leaving GHCi.
```

Task 4 - Recursive List Processors

Code:

rlp.hs

```haskell
list2set [] = []
list2set (x : xy) = if (elem x xy) then (z) else (x : z)
    where z = list2set xy

isPalindrome [] = True
isPalindrome (x : xy) = if (length (x : xy)) == 1 then True
    else (if (x == last xy) then (z) else False)
        where z = isPalindrome (head xy : drop 1 (init xy))

collatzFunc :: Int -> Int
collatzFunc 1 = 1
collatzFunc number = if (even number) then (number `div` 2)
    else ((3 * number) + 1)

collatz :: Int -> [Int]
collatz number = if number == 1 then [1]
    else [number] ++ collatz (collatzFunc number)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l "rlp.hs"
[1 of 1] Compiling Main             ( rlp.hs, interpreted )
Ok, one module loaded.
ghci> :set prompt ">>>"
>>> 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 [1,2,5,7,11,7,5,3,2]
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]
```

Task 5 - List Comprehensions

Code:

lc.hs

```haskell
list2set [] = []
list2set (x : xy) = if (elem x xy) then (z) else (x : z)
    where z = list2set xy

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

freqTable :: Eq a => [a] -> [(a, Int)]
freqTable xs = [(x, count x xs) | x <- list2set xs]
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l "lc.hs"
[1 of 1] Compiling Main             ( lc.hs, interpreted )
Ok, one module loaded.
ghci> :set prompt ">>>"
>>> 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.
```

Task 6 - Higher Order Functions

Code:

hof.hs

```haskell
tgl :: Int -> Int
tgl x = foldl (+) 0 ([1..x])

triangleSequence :: Int -> [Int]
triangleSequence x = map tgl [1..x]

vowelCount :: String -> Int
vowelCount x = length(filter(\y -> y `elem` "aeiou") x)

lcsim :: (a -> b) -> (a -> Bool) -> [a] -> [b]
lcsim f p xs = map f (filter p xs)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :load "hof.hs"
[1 of 1] Compiling Main             ( hof.hs, interpreted )
Ok, one module loaded.
ghci> :set prompt ">>> "
>>> tgl 5
15
>>> tgl 10
55
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> animals = ["elephant", "lion", "tiger", "orangatan", "jaguar"]
>>> lcsim length (\w -> elem (head w) "aeiou") animals
[8,9]
>>> :quit
Leaving GHCi.
```

Task 7 - An Interesting Statistic nPVI

Task 7a - Test Data

Code:

npvi.hs

```haskell
-- variability index -----------------------------------------------
-------------------------------------------------------------------

-- 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]

```

Task 7b - The pairwiseValues Function

Code:

```haskell
-- pairwiseValues function -----------------------------------------
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues list = zip list (tail list)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>>  :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseValues a
[(2,5),(5,1),(1,3)]
>>> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
>>> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
>>> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
>>> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
>>>
```

Task 7c - The pairwiseDifferences Function

Code:

```haskell
-- pairwiseDifferences Function ----------------------------------------
pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences list = map (\(x,y) -> x - y) (pairwiseValues list)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseDifferences a
[-3,4,-2]
>>> pairwiseDifferences b
[-2,-3,4,-3]
>>> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
>>> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
>>> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
>>>
```

Task 7d - The pairwiseSums Function

Code:

```haskell
-- pairwiseSums Function ------------------------------------------------
pairwiseSums :: [Int] -> [Int]
pairwiseSums list = map (\(x,y) -> x + y) (pairwiseValues list)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
>>> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
>>>
```

Task 7e - The pairwiseHalves Function

Code:

```haskell
-- compute half ---------------------------------------
half :: Int -> Double
half list = (fromIntegral number) / 2

-- pairwiseHalves Function ----------------------------
pairwiseHalves :: [Int] -> [Double]
pairwiseHalves list = map half list
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
>>> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
>>> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
>>> _
```

Task 7f - The pairwiseHalfSums Function

Code:

```haskell
42  -- pairwiseHalfSums Function ------------------------------------
43  pairwiseHalfSums :: [Int] -> [Double]
44  pairwiseHalfSums list = pairwiseHalves (pairwiseSums list)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseHalfSums a
[3.5,3.0,2.0]
>>> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
>>> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
>>> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
>>> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

Task 7g - The pairwiseTermPairs Function

Code:

```haskell
-- pairwiseTermPairs Function ----------------------------------------
pairwiseTermPairs :: [Int] -> [(Int,Double)]
pairwiseTermPairs list = zip (pairwiseDifferences list) (pairwiseHalfSums list)

```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
>>> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
>>> pairwiseTermPairs c
[(0,4.0),(2,3.0),(1,1.5),(0,1.0),(-1,1.5),(0,2.0),(-2,3.0),(0,4.0),(-4,6.0)]
>>> pairwiseTermPairs u
[(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0)]
>>> pairwiseTermPairs x
[(-8,5.0),(7,5.5),(-6,5.0),(5,5.5),(-4,5.0),(5,4.5),(-6,5.0),(7,4.5),(-8,5.0)]
>>> _
```

Task 7h - The pairwiseTerms Function

Code:

```haskell
term :: (Int,Double) -> Double
term pair = abs (fromIntegral (fst pair) / (snd pair))

-- pairwiseTerms Function ---------------------------------------
pairwiseTerms :: [Int] -> [Double]
pairwiseTerms list = map term (pairwiseTermPairs list)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
>>> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
>>> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666]
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
>>> _
```

Task 7i - The nPVI Function

Code:

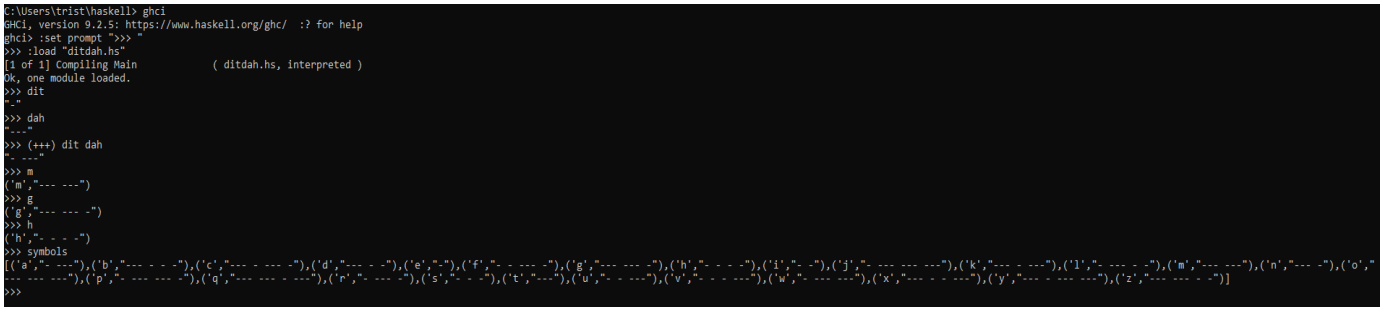
```haskell
-- nPVI Function ------------------------------------------------------
nPVI :: [Int] -> Double
nPVI xs = normalizer xs * sum (pairwiseTerms xs)
    where normalizer xs = 100 / fromIntegral ((length xs) - 1)
```

Demo:

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "npvi.hs"
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
>>> _
```

Task 8 - Historic Code: The Dit Dah Code

Task 8a



Task 8b

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "ditdah.hs"
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> assoc 'm' symbols
('m',"--- ---")
>>> assoc 'a' symbols
('a',"- ---")
>>> find 't'
"---"
>>> find 'z'
"--- --- - -"
```

Task 8c

```
C:\Users\trist\haskell> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "ditdah.hs"
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> addletter "g" "j"
"g   j"
>>> addword "good" "bye"
"good       bye"
>>> droplast3 "water"
"wa"
>>> droplast7 "interesting"
"inte"
```

Task 8d

```
C:\Users\trist\haskell>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "ditdah.hs"
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'e'
"-"
>>> encodeletter 'p'
"- --- --- -"
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "why"
"- --- ---   - - - -   --- - --- ---"
>>> encodeword "okay"
"--- --- ---   --- - ---   - ---   --- - --- ---"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
>>> encodemessage "thank you"
"---   - - - -   - ---   --- -   --- - ---       --- - --- ---   --- --- ---   - - ---"
>>> encodemessage "why not"
"- --- ---   - - - -   --- - --- ---       --- -   --- --- ---   ---"
```