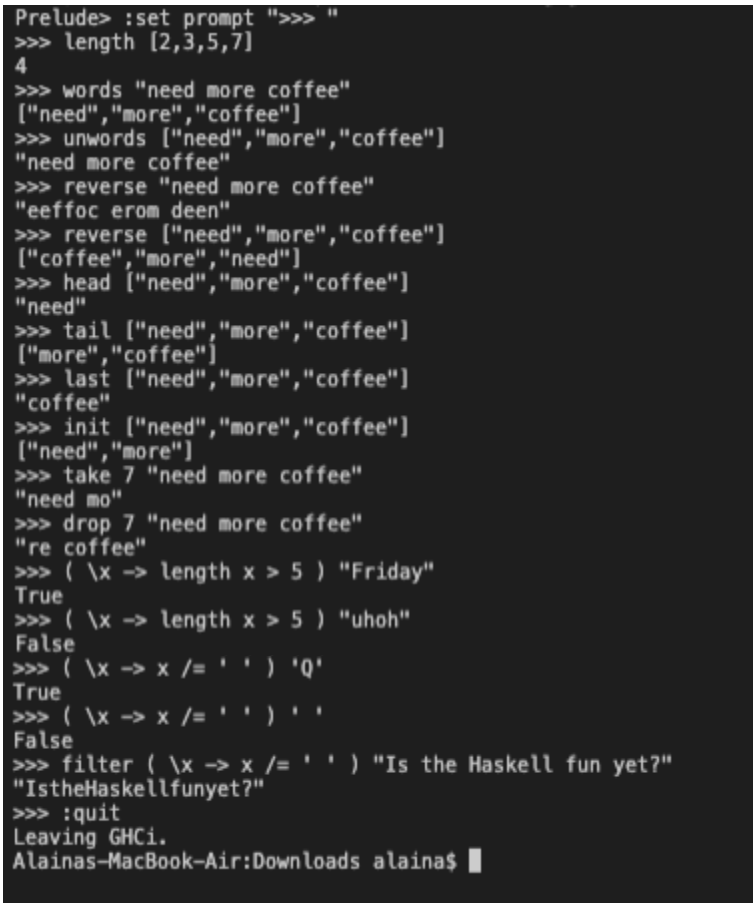
Haskell Programming Assignment Solution

Learning Abstract:

In this assignment I learned more about the syntax and functionality of Haskell. Along with creating numeric and string functions, I was able to learn more about recursive list functions, higher order functions such as zip, map, and foldl, the nPVI, and the famous Dit Dah code through the tasks in this assignment.

Task 1 - Mindfully Mimicking the Demo

Demo:



Task 2 - Numeric Function Definitions

Code:

```
squareArea :: Num a => a -> a | squareArea :: Num a => a -> a
squareArea side = side * side

circleArea :: Floating a => a -> a | circleArea :: Floating a => a -> a
circleArea radius = (radius * radius) * pi

blueAreaOfCube :: Floating a => a -> a | blueAreaOfCube :: Floating a => a -> a
blueAreaOfCube edge = (squareArea edge - circleArea (1/4 * edge)) * 6

paintedCube1 :: (Num p, Ord p) => p -> p | paintedCube1 :: (Num p, Ord p) => p -> p
paintedCube1 order = if (order > 2)  then (((order - 2)^2) * 6) else 0

paintedCube2 :: (Ord p, Num p) => p -> p | paintedCube2 :: (Ord p, Num p) => p -> p
paintedCube2 order = if (order > 2) then ((order - 2) * 12) else 0
```

Demo:

```
*Main> squareArea 10
100
*Main> squareArea 12
144
*Main> circleArea 10
314.1592653589793
*Main> circleArea 12
452.3893421169302
*Main> blueAreaOfCube 10
482.19027549038276
*Main> blueAreaOfCube 12
694.3539967061512
*Main> blueAreaOfCube 1
4.821902754903828
*Main> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
*Main> paintedCube1 1
0
*Main> paintedCube1 2
0
*Main> paintedCube1 3
6
*Main> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
*Main> paintedCube2 1
0
*Main> paintedCube2 2
0
*Main> paintedCube2 3
12
*Main> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
*Main> :quit
Leaving GHCi.
```

Task 3 - Puzzlers

Code:

```
reverseWords :: String -> String | reverseWords :: String -> String
reverseWords charString = unwords (reverse (words charString))

averageWordLength :: Fractional a => [Char] -> a | averageWordLength :: Fractional a => [Char] -> a
averageWordLength charString = fromIntegral nmbrOfLetters / fromIntegral nmbrOfWords
    where nmbrOfLetters = length (filter (\x -> x /= ' ') charString)
          nmbrOfWords = length (words charString)
```

Demo:

```
*Main> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
*Main> reverseWords "want me some coffee"
"coffee some me want"
*Main> reverseWords "the cat is fluffy"
"fluffy is cat the"
*Main> reverseWords "the hamster ate a seed"
"seed a ate hamster the"
*Main> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
*Main> averageWordLength "want me some coffee"
4.0
*Main> averageWordLength "the cat is fluffy"
3.5
*Main> averageWordLength "the hamster ate a seed"
3.6
*Main> :quit
Leaving GHCi.
```

Task 4 - Recursive List Processors

Code:

```
list2set [] = []
list2set (x : xs) = if (elem x xs) then (list2set xs) else (x : list2set xs)

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

collatz :: Int -> [Int]
collatz 1 = [1]
collatz num = if (even num) then num : collatz (num `div` 2) else num : collatz (3 * num + 1)
```

Demo:

```
*Main> list2set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
*Main> list2set "need more coffee"
"ndmr cofe"
*Main> isPalindrome ["coffee","latte","coffee"]
True
*Main> isPalindrome ["coffee","latte","espresso","coffee"]
False
*Main> isPalindrome [1,2,5,7,11,13,11,7,5,3,2]
False
*Main> isPalindrome [2,3,5,7,11,13,11,7,5,3,2]
True
*Main> collatz 10
[10,5,16,8,4,2,1]
*Main> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
*Main> 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]
*Main> :quit
Leaving GHCi.
```

Task 5 - List Comprehensions

Code:

```
count obj objList = length [object | object <- objList, obj == object]
```

Demo:

```
*Main> count 'e' "need more coffee"
5
*Main> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
```

Task 6 - Higher Order Functions

Code:

```
tgl value = foldl (+) 0 [1..value]

triangleSequence value = map tgl [1..value]

lcsim mapFunc filterPred elemList = map mapFunc (filter filterPred elemList)
```

Demo:

```
*Main> tgl 5
15
*Main> tgl 10
55
*Main> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
*Main> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
*Main> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
*Main> animals = ["elephant","lion","tiger","orangatan","jaguar"]
*Main> lcsim length (\w -> elem ( head w ) "aeiou") animals
[8,9]
*Main> []
```

Task 7 - An Interesting Statistic: nPVI

7a - Test data

Code:

```
-- 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]
```

7b - The pairwiseValues function

Code:

```
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues valueList = zip valueList (tail valueList)
```

Demo:

```
*Main> pairwiseValues a
[(2,5),(5,1),(1,3)]
*Main> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
*Main> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
*Main> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
*Main> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
```

7c - The pairwiseDifferences function

Code:

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

Demo:

```
*Main> pairwiseDifferences a
[-3,4,-2]
*Main> pairwiseDifferences b
[-2,-3,4,-3]
*Main> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
*Main> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
*Main> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
```

7d - The pairwiseSums function

Code:

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

Demo:

```
*Main> pairwiseSums a
[7,6,4]
*Main> pairwiseSums b
[4,9,8,7]
*Main> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
*Main> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
*Main> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
```

7e - The pairwiseHalves function

Code:

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

Demo:

```
*Main> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
*Main> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
*Main> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

7f - The pairwiseHalfSums function

Code:

```
pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums valueList = pairwiseHalves(pairwiseSums valueList)
```

Demo:

```
*Main> pairwiseHalfSums a
[3.5,3.0,2.0]
*Main> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
*Main> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
*Main> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
*Main> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

7g - The pairwiseTermPairs function

Code:

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

Demo:

```
*Main> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
*Main> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
*Main> 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)]
*Main> 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)]
*Main> 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)]
```

7h - The pairwiseTerms function

Code:

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

pairwiseTerms valueList = map term (pairwiseTermPairs valueList)
```

Demo:

```
*Main> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
*Main> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
*Main> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666]
*Main> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
*Main> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
```

7i - The nPvi function

Code:

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

Demo:

```
*Main> nPVI a
106.34920634920636
*Main> nPVI b
88.09523809523809
*Main> nPVI c
37.03703703703703
*Main> nPVI u
0.0
*Main> nPVI x
124.98316498316497
```

Task 8 - Historic Code: The Dit Dah Code

Demo: