Haskell Programming Assignment #1: Various Computations

Learning Abstract:

This assignment covers several various use cases for Haskell. There are functions for list processing, math functions and sequences, some play with words and phrases, an advanced math function and some play with Morse Code.

Task #1: Mindfully Mimicking the Demo

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages> 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

```haskell
-----------------------------------
-- Haskell Project 8 --------------
-- Dustin McMahon -----------------
-- 11/27/2022 ---------------------
-----------------------------------

-----------------------------------
-- squareArea :: area of a square
-- 1 param :: side length
squareArea :: Num a => a -> a
squareArea side = side * side

-----------------------------------
-- circleArea :: area of a circle
-- 1 param :: radius length
circleArea :: Fractional f => f -> f
circleArea radius = 3.141592653589793238 * ( radius * radius )

-----------------------------------
-- blueAreaOfCube :: blue area of a cube
---- explained as a cube with 6 blue sides with a white dot
---- dot radius is 1/4 the side
-- 1 param :: side length
blueAreaOfCube :: Fractional f => f -> f
blueAreaOfCube side = sideArea * 6
    where sideArea = squareArea side - circleArea radius
          radius = side / 4

-----------------------------------
-- paintedCube1 :: number of cubes with 1 side painted
---- painted cube disected into an NxNxN cube
-- 1 param :: order of cube (N)
paintedCube1 :: ( Ord a, Num a) => a -> a
paintedCube1 n =
    if (n < 3 ) then 0 else
    sideCount * 6
    where sideCount = ( n - 2 ) * ( n - 2 )

    -----------------------------------
-- paintedCube2 :: number of cubes with 2 side painted
---- painted cube disected into an NxNxN cube
-- 1 param :: order of cube (N)
paintedCube2 :: ( Ord a, Num a ) => a -> a
paintedCube2 n =
    if (n < 3 ) then 0 else
    sideCount * 6
    where sideCount = ( n - 2 ) * 2

```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l 'ha.hs'
target 'ha.hs' is not a module name or a source file
ghci> :l "ha.hs"
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
ghci> squareArea 10
100
ghci> squareArea 12
144
ghci> circleArea 10
314.1592653589793
ghci> circleArea 12
452.3893421169302
ghci> blueAreaOfCube 10
482.19027549038276
ghci> blueAreaOfCube 12
694.3539967061512
ghci> blueAreaOfCube 1
4.821902754903828
ghci> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
ghci> paintedCube1 1
0
ghci> paintedCube1 2
0
ghci> paintedCube1 3
6
ghci> map paintedCube1 [1..10]
[0,0,6,24,54,96,150,216,294,384]
ghci> paintedCube2 1
0
ghci> paintedCube2 2
0
ghci> paintedCube2 3
12
ghci> map paintedCube2 [1..10]
[0,0,12,24,36,48,60,72,84,96]
ghci> :q
Leaving GHCi.
```

Task #3: Puzzlers

```haskell
-----------------------------------
-- reverseWords :: reverse the order of the words in a string
-- 1 param :: character string
reverseWords input = unwords reversed
    where reversed = reverse wordList
          wordList = words input

-----------------------------------
-- averageWordLength :: average length of the words in the phrase
-- 1 param :: character string
averageWordLength input = fromIntegral charTotal / fromIntegral wordCount
    where charTotal = sum lengthMap
          wordCount = length lengthMap
          lengthMap = map ( \x -> length x ) inputList
          inputList = words input

```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l ha
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
ghci> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
ghci> reverseWords "want me some coffee"
"coffee some me want"
ghci> reverseWords "Why do nurses like red crayons?"
"crayons? red like nurses do Why"
ghci> reverseWords "Sometimes they have to draw blood"
"blood draw to have they Sometimes"
ghci> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
ghci> averageWordLength "want me some coffee"
4.0
ghci> averageWordLength "Why do nurses like red crayons?"
4.333333333333333
ghci> averageWordLength "Sometimes they have to draw blood"
4.666666666666667
ghci> :q
Leaving GHCi.
```

Task #4: Recursive List Processors

```haskell
-----------------------------------
-- list2set :: removes any duplicates from the list
-- 1 param :: list of objects
list2set [] = []
list2set theList = headObject : list2set filteredTail
    where filteredTail = filter ( \x -> x /= headObject) tailObject
          headObject = head theList
          tailObject = tail theList

-----------------------------------
-- isPalindrome :: True if reads the same forward as backward
-- 1 param :: list of objects
isPalindrome [] = True
isPalindrome theList =
    if ( length theList == 1 ) then True else
    if ( firstObject == lastObject ) then isPalindrome newList else False
    where newList = init tailList
          tailList = tail theList
          firstObject = head theList
          lastObject = last theList

-----------------------------------
-- collatz :: collatz sequence given x
-- 1 param :: pos number greater than 0
collatz 1 = [1]
collatz x = x : rest
    where rest = if( modX == 0) then collatz smallerVal else collatz largerVal
          modX = x `mod` 2
          smallerVal = x `div` 2
          largerVal = (( 3 * x ) + 1 )

```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l ha
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
ghci> list2set [1,2,3,2,3,4,3,4,5]
[1,2,3,4,5]
ghci> list2set "need more coffee"
"ned morcf"
ghci> isPalindrome ["coffee","latte","coffee"]
True
ghci> isPalindrome ["coffee","latte","espresso","coffee"]
False
ghci> isPalindrome [1,2,5,7,11,13,11,7,5,3,2]
False
ghci> isPalindrome [2,3,5,7,11,13,11,7,5,3,2]
True
ghci> collatz 10
[10,5,16,8,4,2,1]
ghci> collatz 11
[11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
ghci> 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]
ghci> :q
Leaving GHCi.
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8>
```

Task #5: List Comprehensions

```haskell
-------------------------------------
-- count :: count elements occurance within a list
-- 2 param :: element, list of same type as element
count elem theList = length [ x | x <- theList, x == elem ]

-------------------------------------
-- count :: list of pairs [(uniqueElement, count),...]
-- 1 param :: list of elements
freqTable theList = [ ( x, count x theList ) | x <- uniqueList ]
    where uniqueList = list2set theList
```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l ha
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
ghci> count 'e' "need more coffee"
5
ghci> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
ghci> freqTable "need more coffee"
[('n',1),('e',5),('d',1),(' ',2),('m',1),('o',2),('r',1),('c',1),('f',2)]
ghci> 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)]
ghci> :q
Leaving GHCi.
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8>
```

Task #6: Higher Order Functions

```haskell
-----------------------------------
-- tgl :: triangular number of x
-- 1 param :: x (the number)
tgl 1 = 1
tgl x = x + tgl (x-1)

-----------------------------------
-- triangleSequence :: triangular Sequence from 1 to x
-- 1 param :: x (the number)
triangleSequence x = map tgl [1..x]

-----------------------------------
-- vowelCount :: count vowels in a string
-- 1 param :: the string
vowelCount theString = length vowelList
    where vowelList = filter ( \x -> elem x "aeiou" ) theString

-----------------------------------
-- lcsim :: list comprehension simulation
-- 3 param :: [ f x | x <- xs, p x ]
---- map function
---- predicate
---- list
lcsim mapFunc pred theList = [ mapFunc x | x <- theList, pred x ]
```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l ha
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
ghci> tgl 5
15
ghci> tgl 10
55
ghci> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
ghci> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
ghci> vowelCount "cat"
1
ghci> vowelCount "mouse"
3
ghci> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
ghci> animals = ["elephant","lion","tiger","orangatan","jaguar"]
ghci> lcsim length (\w -> elem ( head w ) "aeiou") animals
[8,9]
ghci> :q
Leaving GHCi.
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8>
```

Task #7: An Interesting Statistic: nPVI

```haskell
-----------------------------------
-- nPVI Pairs ---------------------
-- Dustin McMahon -----------------
-- 11/27/2022 ---------------------
-----------------------------------

-- 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 :: pairs a number with the next in the list
-- 1 param :: list of ints
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues list = zip ( init list ) ( tail list )

-----------------------------------
-- pairwiseDifferences :: the difference between the pairwiseValues of the list
-- 1 param :: list of ints
pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences list = map ( \(x,y) -> x - y ) ( pairwiseValues list )

-----------------------------------
-- pairwiseSums :: the sum of the pairwiseValues of the list
-- 1 param :: list of ints
pairwiseSums :: [Int] -> [Int]
pairwiseSums list = map ( \(x,y) -> x + y ) ( pairwiseValues list )

-----------------------------------
-- half :: GIVEN CODE half the given value
-- 1 param :: int value
half :: Int -> Double
half number = ( fromIntegral number ) / 2

-----------------------------------
-- pairwiseHalves :: halve the values of all ints in list
-- 1 param :: list of ints
pairwiseHalves :: [Int] -> [Double]
pairwiseHalves list = map half list
```

```haskell

    -----------------------------------
    -- pairwiseHalfSums :: sum the list then divide by 2
    -- 1 param :: list of ints
    pairwiseHalfSums :: [Int] -> [Double]
    pairwiseHalfSums list = pairwiseHalves ( pairwiseSums list )

    -----------------------------------
    -- pairwiseTermPairs :: list of pairs [(difference,halfSum)...] given a list
    -- 1 param :: list of ints
    pairwiseTermPairs :: [Int] -> [(Int,Double)]
    pairwiseTermPairs list = zip ( pairwiseDifferences list ) ( pairwiseHalfSums list )

    -----------------------------------
    -- term :: GIVEN CODE term value given pair (a,b) = absolute(a/b)
    -- 1 param :: int value
    term :: (Int,Double) -> Double
    term ndPair = abs ( fromIntegral ( fst ndPair ) / ( snd ndPair ) )

    -----------------------------------
    -- pairwiseTerms :: list of terms give a list
    -- 1 param :: list of ints
    pairwiseTerms :: [Int] -> [Double]
    pairwiseTerms list = map term ( pairwiseTermPairs list )

    -----------------------------------
    -- nPVI :: GIVEN CODE term value given pair (a,b) = absolute(a/b)
    -- 1 param :: int value
    nPVI :: [Int] -> Double
    nPVI xs = normalizer xs * sum ( pairwiseTerms xs )
        where normalizer xs = 100 / fromIntegral ( ( length xs ) - 1 )
```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l npvi
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
ghci> pairwiseValues a
[(2,5),(5,1),(1,3)]
ghci> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
ghci> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
ghci> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
ghci> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
ghci>
```

```
PS C:\Users\dpmcm\Desktop\SUNY Oswego\CSC344\ProgLanguages\project8> ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :l npvi
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
ghci> pairwiseDifferences a
[-3,4,-2]
ghci> pairwiseDifferences b
[-2,-3,4,-3]
ghci> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
ghci> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
ghci> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
ghci> pairwiseSums a
[7,6,4]
ghci> pairwiseSums b
[4,9,8,7]
ghci> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
ghci> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
ghci> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
ghci> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
ghci> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
ghci> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
ghci>
```

```
ghci> :r
Ok, one module loaded.
ghci> pairwiseHalfSums a
[3.5,3.0,2.0]
ghci> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
ghci> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
ghci> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
ghci> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
ghci> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
ghci> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
ghci> 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)]
ghci> 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)]
ghci> 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)]
ghci> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
ghci> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
ghci> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666666666666]
ghci> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
ghci> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
ghci> nPVI a
106.34920634920636
ghci> nPVI b
88.09523809523809
ghci> nPVI c
37.03703703703703
ghci> nPVI u
0.0
ghci> nPVI x
124.98316498316497
ghci>
```

Task #8: Historic Code: The Dit Dah Code

```
ghci> :l ditdah
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
ghci> dit
"-"
ghci> dah
"---"
ghci> dit +++ dah
"- ---"
ghci> m
('m',"--- ---")
ghci> g
('g',"--- --- -")
ghci> h
('h',"- - - -")
ghci> symbols
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -"),('d',"--- - -"),('e',"-"),('f',"- - --- -"),('g',"--- --- -"),
 - ---"),('r',"- --- -"),('s',"- - -"),('t',"---"),('u',"- - ---"),('v',"- - - ---"),('w',"- --- ---"),('x',"--- - -
ghci>
```

```
ghci> assoc 'e' symbols
('e',"-")
ghci> assoc 'd' symbols
('d',"--- - -")
ghci> find 'a'
"- ---"
ghci> find 'x'
"--- - - ---"
ghci>
```

```
ghci> addletter "a" "b"
"a   b"
ghci> addword "bob" "uncle"
"bob       uncle"
ghci> droplast3 "bob is my uncle"
"bob is my un"
ghci> droplast7 "bob is my uncle"
"bob is m"
ghci>
```

```
ghci> encodeletter 'm'
"--- ---"
ghci> encodeletter 'd'
"--- - -"
ghci> encodeletter 'c'
"--- - --- -"
ghci> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
ghci> encodeword "dustin"
"--- - -   - - ---   - - -   ---   - -   --- -"
ghci> encodeword "code"
"--- - --- -   --- --- ---   --- - -   -"
ghci> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
ghci> encodemessage "i love code"
"- -       - --- - -   --- --- ---   - - - ---   -       --- - --- -   --- --- ---   --- - -   -"
ghci> encodemessage "project done"
"- --- --- -   - --- -   --- --- ---   - --- --- ---   -   --- - --- -   ---       --- - -   --- --- ---   --- -   -"
ghci>
```