Haskell Programming Assignment: Various Computions

Learning Abstract: This programming exercise focuses on Haskell functions, list comprehensions, recursive list processing, and higher order functions.

The eight key activities are completed by using GHCi and a suitable text editor.

We also have to create a document with a clean structure that includes examples of all 8 of the tasks.

Task 1 - Mindfully Mimicking the Demo

```
C:\Users\Erica\Downloads\DEV>ghci
GHCi, version 9.4.3: 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’

<interactive>:15:15: error: lexical error at character ' '
>>> ( \x -> x /= ' ' ) 'Q'
True
>>>( \x -> x /= ' ' ) ' '
False
>>>filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>>:quit
Leaving GHCi.
```

Task 2 - Numeric Function Definitions

```
squareArea num = num * num
circleArea radius = pi * radius^2
blueAreaOfCube x = 6 * ((squareArea x) - (circleArea (x / 4)))
paintedCube1 int = if(int > 2) then (6 * ((int - 2)^2)) else 0
paintedCube2 int = if(int > 2) then (6 * (2 * (int - 2))) else 0
```

```
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]
[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

```haskell
reverseWords str = unwords (reverse(words str))
averageWordLength str = fromIntegral (strLength - spaceN) / fromIntegral nWords
   where
    strLength = length str
    nWords = length (words str)
    spaceN = nWords - 1
```

```
[1 of 2] Compiling Main             ( NFDD.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

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

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

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

```
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 [1,2,5,7,11,13,11,7,5,3,2]
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.
```

Task 5 - List Comprehensions

```haskell
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 ]
```

```
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

```haskell
tgl :: Int -> Int
tgl n = foldl (+) 0 ([1..n])
triangleSequence :: Int -> [Int]
triangleSequence n = map tgl [1..n]
vowelCount :: String -> Int
vowelCount num = length (filter (\n -> n `elem` "aeiou") num)
lcsim :: ( a -> b)->(a -> Bool) -> [a] -> [b]
lcsim l m x = map l (filter m x)
```

```
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]
```

Task 7 - An Interesting Statistic: nPVI

```haskell
---Task 7a---
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---
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues numL = zip numL (tail numL)
---Task 7c---
pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences numL = map (\ (a,b) -> a - b) (pairwiseValues numL)
---Task 7d---
pairwiseSums :: [Int] -> [Int]
pairwiseSums numL = map (\ (a,b) -> a + b) (pairwiseValues numL)
---Task 7e---
half :: Int -> Double
half n = (fromIntegral n) / 2
pairwiseHalves :: [Int] -> [Double]
pairwiseHalves numL = map half numL
```

```haskell
---Task 7f---
pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums numL = pairwiseHalves (pairwiseSums numL)
---Task 7g---
pairwiseTermPairs :: [Int] -> [(Int,Double)]
pairwiseTermPairs numL = zip (pairwiseDifferences numL) (pairwiseHalfSums numL)
---Task 7h---
term :: (Int, Double) -> Double
term f = abs (fromIntegral (fst f) / (snd f))
pairwiseTerms :: [Int] -> [Double]
pairwiseTerms numL = map term (pairwiseTermPairs numL)
---Task 7i---
nPVI :: [Int] -> Double
nPVI xs = normalizer xs * sum ( pairwiseTerms xs )
    where normalizer xs = 100 / fromIntegral ( ( length xs ) - 1 )
```

Task 7a - Test data

```
ghci> a
[2,5,1,3]
ghci> b
[1,3,6,2,5]
ghci> c
[4,4,2,1,1,2,2,4,4,8]
ghci> u
[2,2,2,2,2,2,2,2,2,2]
ghci> x
[1,9,2,8,3,7,2,8,1,9]
ghci> :q
Leaving GHCi.
```

Task 7b - The pairwiseValues function

```
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

```
Ok, one module loaded.
ghci> :set prompt ">>> "
>>> 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

```
Ok, one module loaded.
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]
```

Task 7e - The pairwiseHalves function

```
Ok, one module loaded.
ghci>  :set prompt ">>> "
>>>  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

```
Ok, one module loaded.
ghci> :set prompt ">>> "
>>>  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

```
Ok, one module loaded.
ghci>  :set prompt ">>> "
>>>  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

```
Ok, one module loaded.
ghci> :set prompt ">>> "
>>> 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

```
ghci> :set prompt ">>> "
>>>  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

Subtask 8a

```
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',"--- --- -"),('h',"- - - -"),('i',"- -"),('j
',"- --- --- ---"),('k',"--- - ---"),('l',"- --- - -"),('m',"--- ---"),('n',"--- -"),('o',"--- --- --- ---"),('p',"- --- --- -"),('q',"--- --- - ---")
,('r',"- --- -"),('s',"- - -"),('t',"---"),('u',"- - ---"),('v',"- - - ---"),('w',"- --- ---"),('x',"--- - - ---"),('y',"--- - --- ---"),('z',"---
 --- - -")]
```

Subtask 8b

```
ghci> find 'c'
"--- - --- -"
ghci> find 'z'
"--- --- - -"
ghci> :reload
Ok, one module loaded.
ghci> assoc 'c' symbols
('c',"--- - --- -")
ghci> assoc 'h' symbols
('h',"- - - -")
ghci> find 'z'
"--- --- - -"
ghci> find 'o'
"--- --- ---"
```

Subtask 8c

```
ghci> addletter "l" "s"
"l   s"
ghci> addword "big" "cat"
"big       cat"
ghci> droplast3 "Mathematics"
"Mathemat"
ghci> droplast7 "New York City"
"New Yo"
```

Subtask 8d

```
ghci> encodeletter 'm'
"--- ---"
ghci> encodeletter 'q'
"--- --- - ---"
ghci> encodeletter 'r'
"- --- -"
ghci> encodeword 'yay'

<interactive>:24:12: error:
    * Syntax error on 'yay'
      Perhaps you intended to use TemplateHaskell or TemplateHaskellQuotes
    * In the Template Haskell quotation 'yay'
ghci> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
ghci> encodeword "twelve"
"---   - --- ---   -   - --- - -   - - - ---   -"
ghci> encodeword "you"
"--- - --- ---   --- --- ---   - - ---"
ghci> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
ghci> encodemessage "i like potatoes"
"- -       - --- - -   - -   --- - ---   -       - --- --- -   --- --- ---   ---   - ---   ---   --- --- ---   -   - - -"
ghci> encodemessage "my cats name is danii"
"--- ---   --- - --- ---       --- - --- -   - ---   ---   - - -       --- -   - ---   --- ---   -       - -   - - -       --- - -   - ---   --- -
   - -   - -"
```