Haskell Programming Assignment #1: Various Computations

Brandon LaPointe

Learning Abstract

This assignment contains programming exercises that focus on functions, recursive list processing, list comprehensions, and higher order functions in Haskell.

Task 1: Mindfully Mimicking the Demo

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> 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

```
-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 2 -------------------------------------------------------------------------------
-------------------- Numeric Function Definitions ---------------------------------------------------------------------

-------------------------------------------------------------------------
-- squareArea

squareArea :: Float -> Float

squareArea x = x * x

-------------------------------------------------------------------------
-- circleArea

circleArea :: Float -> Float

circleArea r = pi * ( r * r )

-------------------------------------------------------------------------
-- blueAreaOfCube

blueAreaOfCube :: Float -> Float

blueAreaOfCube s = 6 * ( squareArea( s ) - circleArea( s / 4 ) )

-------------------------------------------------------------------------
-- paintedCube1

paintedCube1 :: Int -> Int

paintedCube1 n = if ( n <= 2 ) then 0 else 6 * ( ( n - 2 ) * ( n - 2 ) )

-------------------------------------------------------------------------
-- paintedCube2

paintedCube2 :: Int -> Int

paintedCube2 n = if ( n <= 2 ) then 0 else ( n - 2 ) * 12
```

Task 2: Numeric Function Definitions Demo

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

```
-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 3 -------------------------------------------------------------------------------------
--------------------------- Puzzlers ------------------------------------------------------------------------------------

-----------------------------------------------------------------------
-- reverseWords

reverseWords :: String -> String

reverseWords = unwords . reverse . words

-----------------------------------------------------------------------
-- totalChars

totalChars :: String -> Int

totalChars = sum. map length. words

-----------------------------------------------------------------------
-- wordCount

wordCount :: String -> Int

wordCount = length. words

-----------------------------------------------------------------------
-- averageWordLength

averageWordLength :: String -> Double

averageWordLength str = fromIntegral x1 / fromIntegral x2
    where x1 = totalChars str
          x2 = wordCount str
```

Task 3: Puzzlers Demo

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

C:\Users\bmlgu>
```

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "vc.hs"
[1 of 1] Compiling Main             ( vc.hs, interpreted )
Ok, one module loaded.
>>> reverseWords "was it a car or a cat i saw"
"saw i cat a or car a it was"
>>> reverseWords "blessed are they that believe that they are blessed"
"blessed are they that believe that they are blessed"
>>> averageWordLength "was it a car or a cat i saw"
2.111111111111111
>>> averageWordLength "blessed are they that believe that they are blessed"
4.777777777777778
>>> :quit
Leaving GHCi.
```

Task 4: Recursive List Processors Code

```
-------------------------------------------------------------------------------------------------------------------------
---------------------------- Task 4 -----------------------------------------------------------------------------------
-------------------- Recursive List Processors ------------------------------------------------------------------------

-----------------------------------------------------------------------
-- list2set

list2set :: Eq a => [a] -> [a]

list2set [] = []
list2set (x:xs)
    | x `elem` xs = list2set xs
    | otherwise   = x : list2set xs


-----------------------------------------------------------------------
-- isPalindrome

isPalindrome :: Eq a => [a] -> Bool

isPalindrome []  = True
isPalindrome [_] = True
isPalindrome xs  = (head xs == last xs) && (isPalindrome (tail (init xs)))

-----------------------------------------------------------------------
-- collatzNum

collatzNum :: Int -> Int

collatzNum 1 = 1
collatzNum n =
    if (even n) then (n `div` 2)
    else (3 * n + 1)

-----------------------------------------------------------------------
-- collatz

collatz :: Int -> [Int]

collatz n =
    if n == 1 then [1]
    else [n] ++ collatz (collatzNum n)
```

Task 4: Recursive List Processors Demo

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "vc.hs"
[1 of 1] Compiling Main             ( vc.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.
```

Task 5: List Comprehension Code

```
-------------------------------------------------------------------------------------------------------------------------
---------------------------- Task 5 -----------------------------------------------------------------------------------
----------------------- List Comprehension ----------------------------------------------------------------------------

-----------------------------------------------------------------------
-- count ( non list comprehensive )
--
-- count :: Eq a => a -> [a] -> Int
--
-- count x = length . filter ( == x )
--

-----------------------------------------------------------------------
-- count ( list comprehensive )

count :: Eq a => a -> [a] -> Int

count x [] = 0
count x xlist = length xs
    where xs = [ xs | xs <- xlist, xs == x ]

-----------------------------------------------------------------------
-- freqTable ( list comprehensive )

freqTable :: Eq a => [a] -> [(a, Int)]

freqTable xlist = [ (x, y) | x <- list2set xlist, let y = (count x xlist)]
```

Task 5: List Comprehension Demo

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "vc.hs"
[1 of 1] Compiling Main             ( vc.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.
```

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "vc.hs"
[1 of 1] Compiling Main             ( vc.hs, interpreted )
Ok, one module loaded.
>>> count 'a' "new car, caviar, four star daydream, think i'll buy me a football team"
9
>>> count 7 [1,4,6,5,7,6,5,4,7,8,5,3,7,3,2,1]
3
>>> freqTable "money, it's a crime"
[('o',1),('n',1),('y',1),(',',1),('t',1),('\'',1),('s',1),('a',1),(' ',3),('c',1),('r',1),('i',2),('m',2),('e',2)]
>>> freqTable [1,4,6,5,7,6,5,4,7,8,5,3,7,3,2,1]
[(6,2),(4,2),(8,1),(5,3),(7,3),(3,2),(2,1),(1,2)]
>>> :quit
Leaving GHCi.
```

Task 6: Higher Order Functions Code

```
-------------------------------------------------------------------------------------------
---------------------------- Task 6 -------------------------------------------------------
---------------------- Higher Order Functions -------------------------------------------

-----------------------------------------------------------------------
-- tgl ( recursive )
--
-- tgl :: Int -> Int
--
-- tgl 0 = 0
-- tgl n = n + ( tgl ( n - 1 ) )

-----------------------------------------------------------------------
-- tgl ( higher order )

tgl :: Int -> Int

tgl 0 = 0
tgl n = foldl (+) 0 [ x | x <- [1..n] ]

-----------------------------------------------------------------------
-- triangleSequence ( higher order )

triangleSequence :: Int -> [Int]

trinagleSequence 0 = [0]
triangleSequence n = map tgl [ x | x <- [1..n] ]

-----------------------------------------------------------------------
-- isVowel ( helper function for vowelCount )

isVowel :: Char -> Bool

isVowel c = if ( c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ) then True
            else False

-----------------------------------------------------------------------
-- vowelCount ( higher order )

vowelCount :: String -> Int

vowelCount s = length $ filter ( \x -> isVowel x ) s

-- vowelCount s = length filter ( \x -> x `elem` "aeiou" ) s

-----------------------------------------------------------------------
-- lcsim ( higher order )

lcsim :: Eq a => (a -> b) -> (a -> Bool) -> [a] -> [b]

lcsim fnct fltr xlist = map fnct xs
    where xs = filter fltr xlist


-------------------------------------------------------------------------------------------
```

Task 6: Higher Order Functions Demo

```
C:\Users\bmlgu>ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :load "vc.hs"
[1 of 1] Compiling Main             ( vc.hs, interpreted )
Ok, one module loaded.
>>> 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]
>>> tgl 7
28
>>> tgl 19
190
>>> triangleSequence 9
[1,3,6,10,15,21,28,36,45]
>>> triangleSequence 17
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153]
>>> vowelCount "mercedesbenz"
4
>>> vowelCount "fluxcapacitor"
5
>>> lcsim tgl even [1..10]
[3,10,21,36,55]
>>> cars = ["mercedesbenz","holden","amc","renault","subaru","porsche","bmw","maserati","toyota","nissan"]
>>> lcsim length (\w -> elem ( head w ) "aeiou" ) cars
[3]
>>>
```

Task 7: nPVI – Code

```
--- Name: Brandon LaPointe ---------------------------------------------------------------------------------------------------------------
--- Course: CSC344 -----------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------
--- nPVI (npvi.hs) contains several pairwise functions leading up to
--- the nPVI function. Normalized Pairwise Variability Index is a
--- measure of the pairwise variability of terms in a sequence of
--- numeric terms.

------------------------------------------------------------------------------------------------------------------------------------------
----------------------------- Task 7a ------------------------------------------------------------------------------------------------------
----------------------------- Test Data ----------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- 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 ------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- pairwiseValues

pairwiseValues :: [Int] -> [(Int,Int)]

pairwiseValues ilist = zip (init ilist) (tail ilist)



------------------------------------------------------------------------------------------------------------------------------------------
----------------------------- Task 7c ------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- pairwiseDifferences

pairwiseDifferences :: [Int] -> [Int]

pairwiseDifferences ilist = map (\(x,y) -> x - y) (pairwiseValues ilist)



------------------------------------------------------------------------------------------------------------------------------------------
----------------------------- Task 7d ------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- pairwiseSums

pairwiseSums :: [Int] -> [Int]

pairwiseSums ilist = map (\(x,y) -> x + y) (pairwiseValues ilist)
```

```
-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 7e ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- half

half :: Int -> Double

half number = ( fromIntegral number ) / 2

-------------------------------------------------------------------------
-- pairwiseHalves

pairwiseHalves :: [Int] -> [Double]

pairwiseHalves ilist = map half ilist


-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 7f ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- pairwiseHalfSums

pairwiseHalfSums :: [Int] -> [Double]

pairwiseHalfSums ilist = map half (pairwiseSums ilist)


-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 7g ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- pairwiseTermPairs

pairwiseTermPairs :: [Int] -> [(Int,Double)]

pairwiseTermPairs ilist = zip (pairwiseDifferences ilist) (pairwiseHalfSums ilist)


-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 7h ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- term

term :: (Int,Double) -> Double

term ndPair = abs ( fromIntegral ( fst ndPair ) / ( snd ndPair ) )

-------------------------------------------------------------------------
-- pairwiseTerms

pairwiseTerms :: [Int] -> [Double]

pairwiseTerms ilist = map term (pairwiseTermPairs ilist)


-------------------------------------------------------------------------------------------------------------------------
------------------------------ Task 7i ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------
-- nPVI

nPVI :: [Int] -> Double

nPVI xs = normalizer xs * sum ( pairwiseTerms xs )
    where normalizer xs = 100 / fromIntegral ( ( length xs ) - 1 )


-------------------------------------------------------------------------------------------------------------------------
```

Task 7a: nPVI – Test Data

```
C:\Users\bmlgu>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.
>>> a
[2,5,1,3]
>>> b
[1,3,6,2,5]
>>> c
[4,4,2,1,1,2,2,4,4,8]
>>> u
[2,2,2,2,2,2,2,2,2,2]
>>> x
[1,9,2,8,3,7,2,8,1,9]
>>>
```

Task 7b: nPVI – pairwiseValues Function

```
C:\Users\bmlgu>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: nPVI – pairwiseDifferences Function

```
C:\Users\bmlgu>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: nPVI – pairwiseSums Function

```
C:\Users\bmlgu>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: nPVI – pairwiseHalves Function

```
C:\Users\bmlgu>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: nPVI – pairwiseHalfSums Function

```
C:\Users\bmlgu>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: nPVI – pairwiseTermPairs Function

```
C:\Users\bmlgu>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: nPVI – pairwiseTerms Function

```
C:\Users\bmlgu>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: nPVI – The nPVI Function

```
C:\Users\bmlgu>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
>>> :quit
Leaving GHCi.
```

Task 8: Historic Code – The Dit Dah Code – Subtask 8a

```
C:\Users\bmlgu>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.
>>> dit
"-"
>>> dah
"---"
>>> ( +++ dit dah )

<interactive>:5:7: error:
    * Couldn't match expected type: String -> [Char]
                  with actual type: [Char]
    * The function `dit' is applied to one value argument,
        but its type `[Char]' has none
      In the expression: dit dah
      In the expression: +++ dit dah
>>> +++

<interactive>:6:1: error: parse error on input `+++'
>>> dit +++ dah
"- ---"
>>> m
('m',"--- ---")
>>> g
('g',"--- --- -")
>>> h
('h',"- - - -")
>>> 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',"--- --- - -")]
>>>
```

Task 8: Historic Code – The Dit Dah Code – Subtask 8b

```
C:\Users\bmlgu>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 'a' symbols
('a',"- ---")
>>> assoc 'z' symbols
('z',"--- --- - -")
>>> find 'b'
"--- - - -"
>>> find 'l'
"- --- - -"
>>>
```

Task 8: Historic Code – The Dit Dah Code – Subtask 8c

```
C:\Users\bmlgu>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 "h" "i"
"h   i"
>>> addword "h" "i"
"h       i"
>>> droplast3 "hello"
"he"
>>> droplast7 "brandonlapointe"
"brandonl"
>>> _
```

Task 8: Historic Code – The Dit Dah Code – Subtask 8d

```
>>> addletter "abba"

<interactive>:31:1: error:
    * No instance for (Show ([Char] -> [Char]))
        arising from a use of `print'
        (maybe you haven't applied a function to enough arguments?)
    * In a stmt of an interactive GHCi command: print it
>>> addletter "ab"

<interactive>:32:1: error:
    * No instance for (Show ([Char] -> [Char]))
        arising from a use of `print'
        (maybe you haven't applied a function to enough arguments?)
    * In a stmt of an interactive GHCi command: print it
>>> addletter "a" "b"
"a   b"
>>> addword "a" "b"
"a       b"
>>> droplast3 "hello"
"he"
>>> droplast7 "brandonlapointe"
"brandonl"
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'a'
"- ---"
>>> encodeletter 'g'
"--- --- -"
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "finally"
"- - --- -   - -   --- -   - ---   - --- - -   - --- - -   --- - --- ---"
>>> encodeword "done"
"--- - -   --- --- ---   --- -   -"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - -
--- -   -   -"
>>> encodemessage "fresh pots"
"- - --- -   - --- -   -   - - -   - - - -       - --- --- -   --- --- ---   ---   - - -"
>>> encodemessage "i dont have a coffee problem"
"- -       --- - -   --- --- ---   --- -   ---       - - - -   - ---   - - - ---   -       - ---       --- - --- -
--- --- ---   - - --- -   - - --- -   -   -       - --- --- -   - --- -   --- --- ---   --- - - -   - --- - -   -
-- ---"
>>> _
```