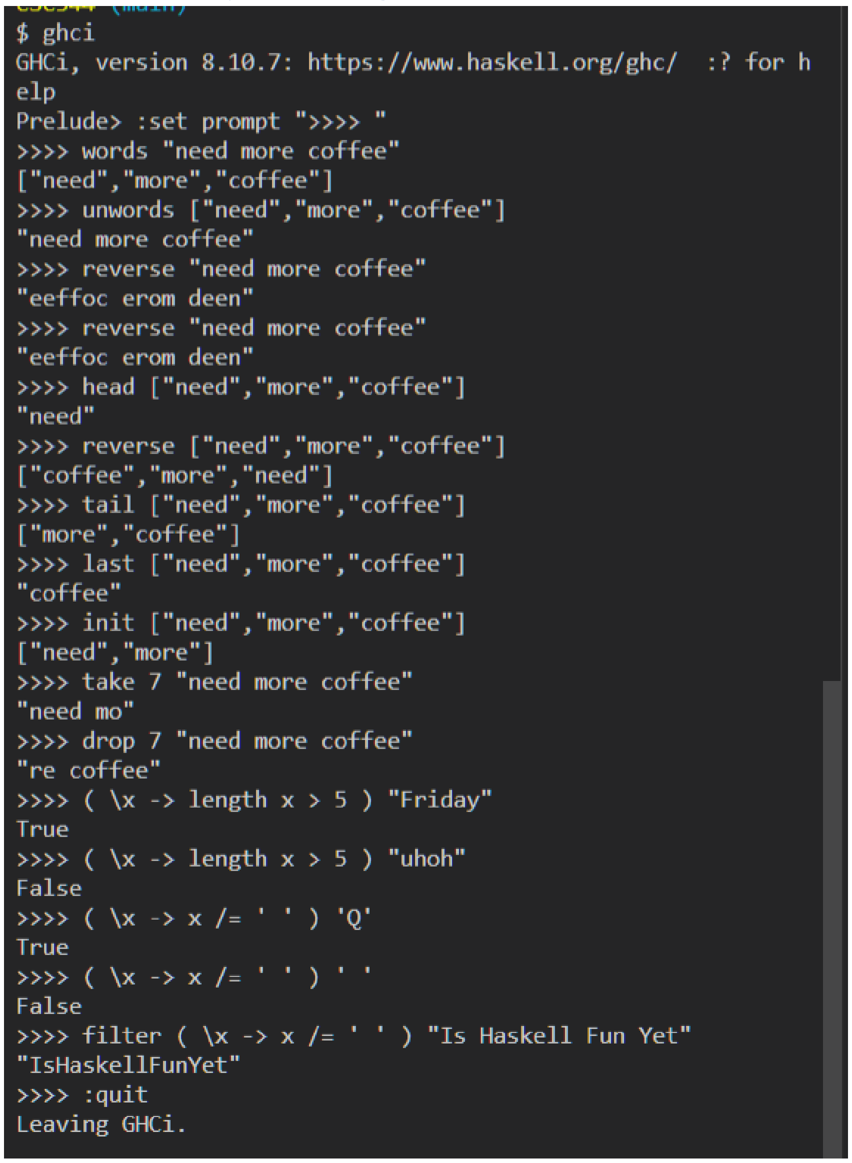
Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Learning Abstract:

This assignment involves various computations in the functional programming language Haskell. It includes making use of common programming paradigms such as list comprehensions, recursion, string processing and high order functions. It also includes a program that encodes phrases to classical morse code and a breakdown of a complex statistics formula through functional means. A slight modification I made after the initial upload was using “guards” instead of if/else/then which helps with readability and conciseness.

Task 1: Mindfully Mimicking the Demo

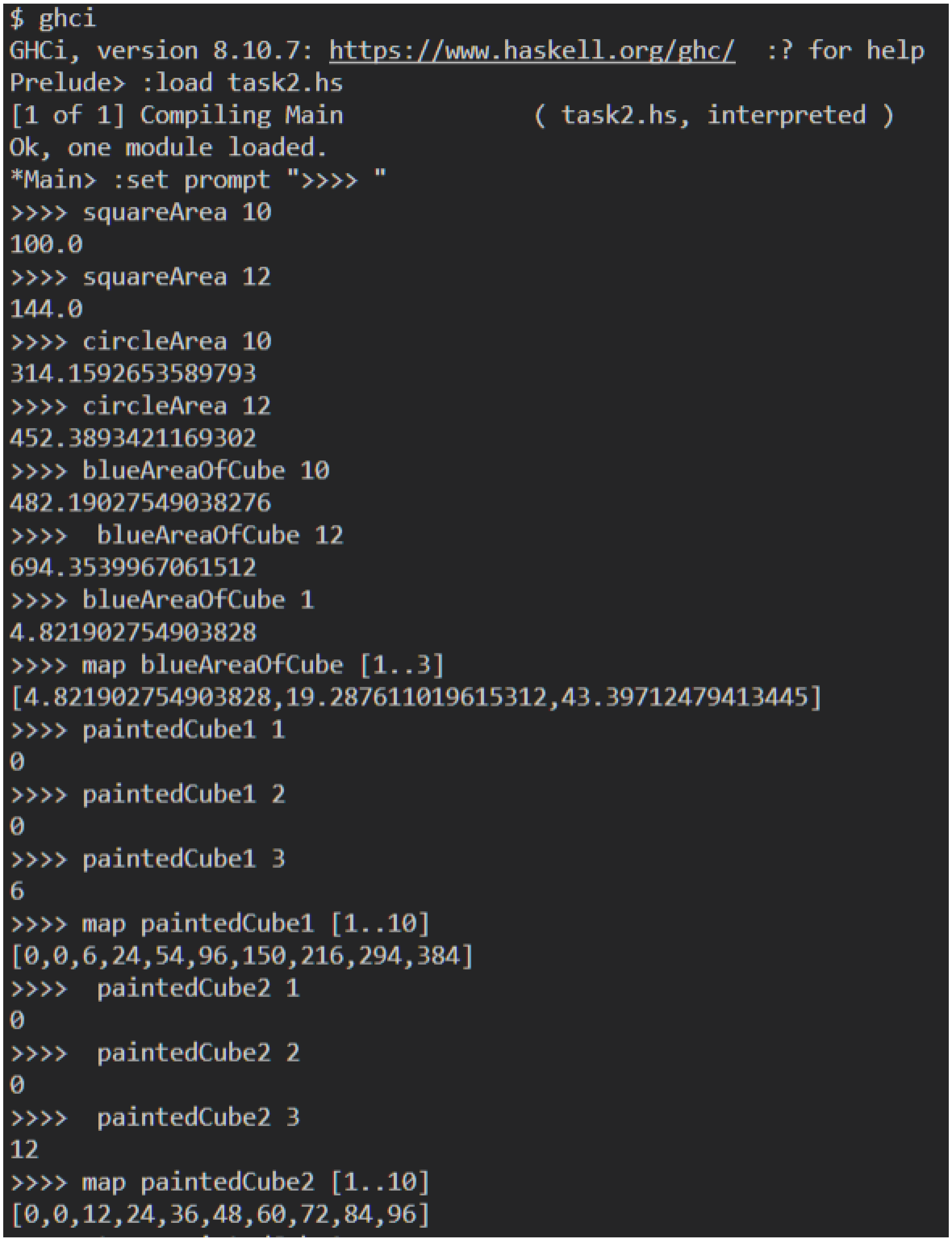


```
$ ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for h
elp
Prelude> :set prompt ">>>> "
>>>> words "need more coffee"
["need","more","coffee"]
>>>> unwords ["need","more","coffee"]
"need more coffee"
>>>> reverse "need more coffee"
"eeffoc erom deen"
>>>> reverse "need more coffee"
"eeffoc erom deen"
>>>> head ["need","more","coffee"]
"need"
>>>> reverse ["need","more","coffee"]
["coffee","more","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 Haskell Fun Yet"
"IsHaskellFunYet"
>>>> :quit
Leaving GHCi.
```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Task 2 - Numeric Function Definitions

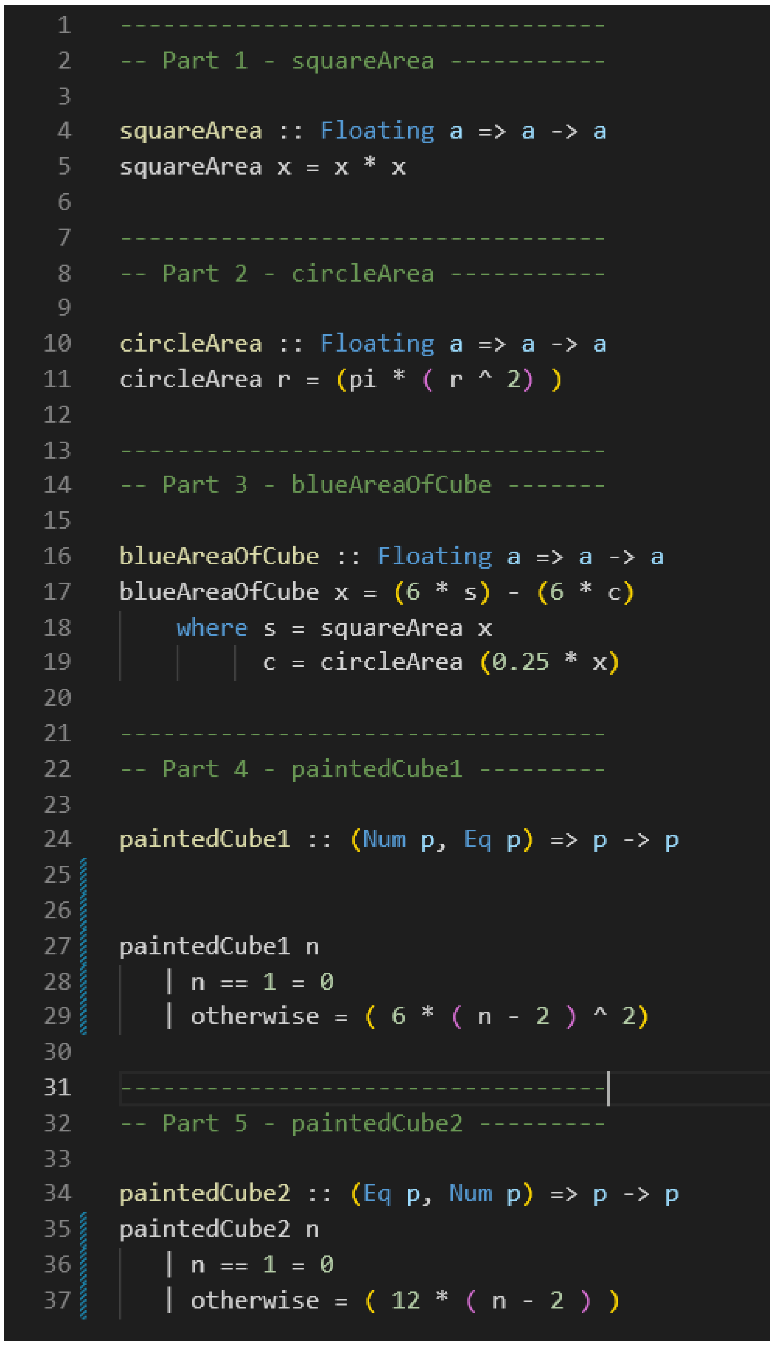
Demo



```
$ ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :load task2.hs
[1 of 1] Compiling Main             ( task2.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>>> "
>>>> squareArea 10
100.0
>>>> squareArea 12
144.0
>>>> 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]
```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Code

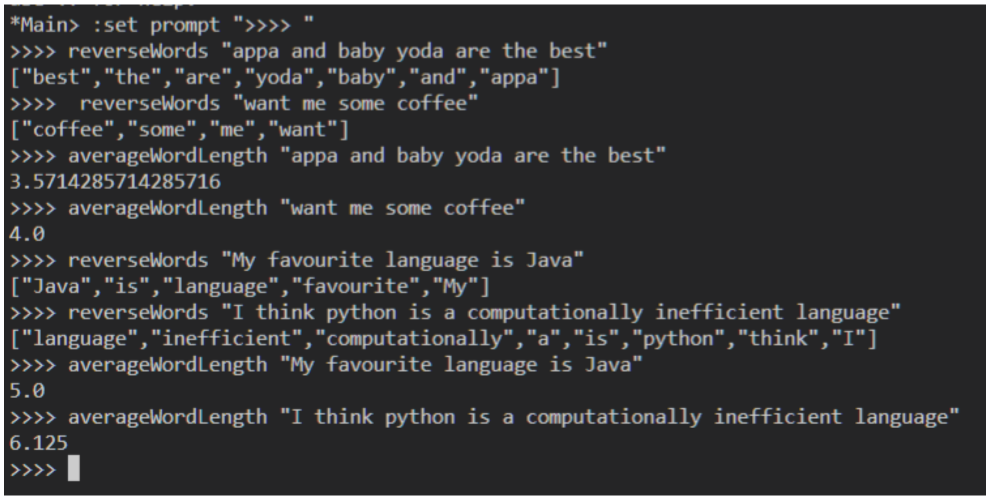


```haskell
----------------------------------
-- Part 1 - squareArea -----------

squareArea :: Floating a => a -> a
squareArea x = x * x

----------------------------------
-- Part 2 - circleArea -----------

circleArea :: Floating a => a -> a
circleArea r = (pi * ( r ^ 2) )

----------------------------------
-- Part 3 - blueAreaOfCube -------

blueAreaOfCube :: Floating a => a -> a
blueAreaOfCube x = (6 * s) - (6 * c)
    where s = squareArea x
          c = circleArea (0.25 * x)

----------------------------------
-- Part 4 - paintedCube1 ---------

paintedCube1 :: (Num p, Eq p) => p -> p


paintedCube1 n
   | n == 1 = 0
   | otherwise = ( 6 * ( n - 2 ) ^ 2)

----------------------------------
-- Part 5 - paintedCube2 ---------

paintedCube2 :: (Eq p, Num p) => p -> p
paintedCube2 n
   | n == 1 = 0
   | otherwise = ( 12 * ( n - 2 ) )
```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

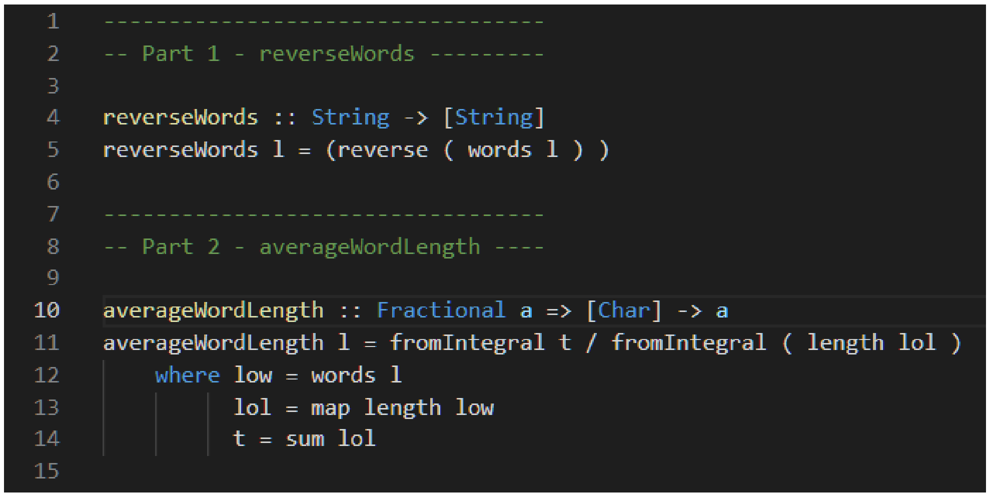
Task 3 – Puzzlers

Demo



```
*Main> :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
>>>> reverseWords "My favourite language is Java"
["Java","is","language","favourite","My"]
>>>> reverseWords "I think python is a computationally inefficient language"
["language","inefficient","computationally","a","is","python","think","I"]
>>>> averageWordLength "My favourite language is Java"
5.0
>>>> averageWordLength "I think python is a computationally inefficient language"
6.125
>>>> 
```

Code

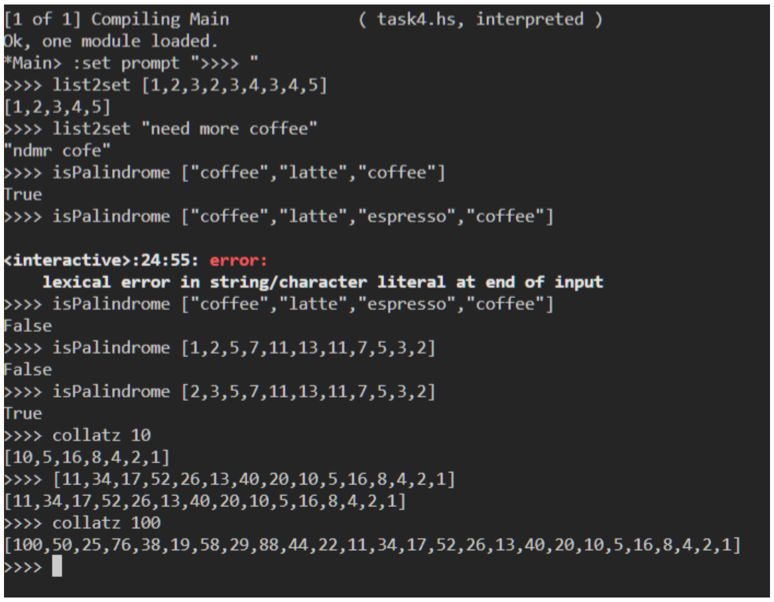


```haskell
----------------------------------
-- Part 1 - reverseWords ---------

reverseWords :: String -> [String]
reverseWords l = (reverse ( words l ) )

----------------------------------
-- Part 2 - averageWordLength ----

averageWordLength :: Fractional a => [Char] -> a
averageWordLength l = fromIntegral t / fromIntegral ( length lol )
    where low = words l
          lol = map length low
          t = sum lol

```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

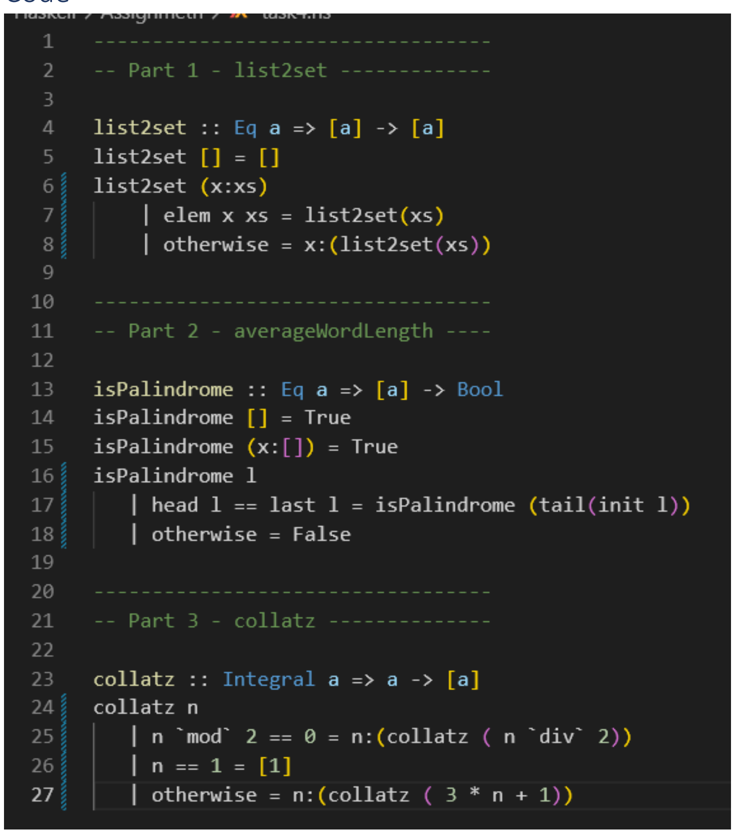
Task 4 -- Recursive List Processors

Demo



```
[1 of 1] Compiling Main             ( task4.hs, interpreted )
Ok, one module loaded.
*Main> :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"]

<interactive>:24:55: error:
    lexical error in string/character literal at end of input
>>>> 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]
>>>> [11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
[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]
>>>>
```

Code

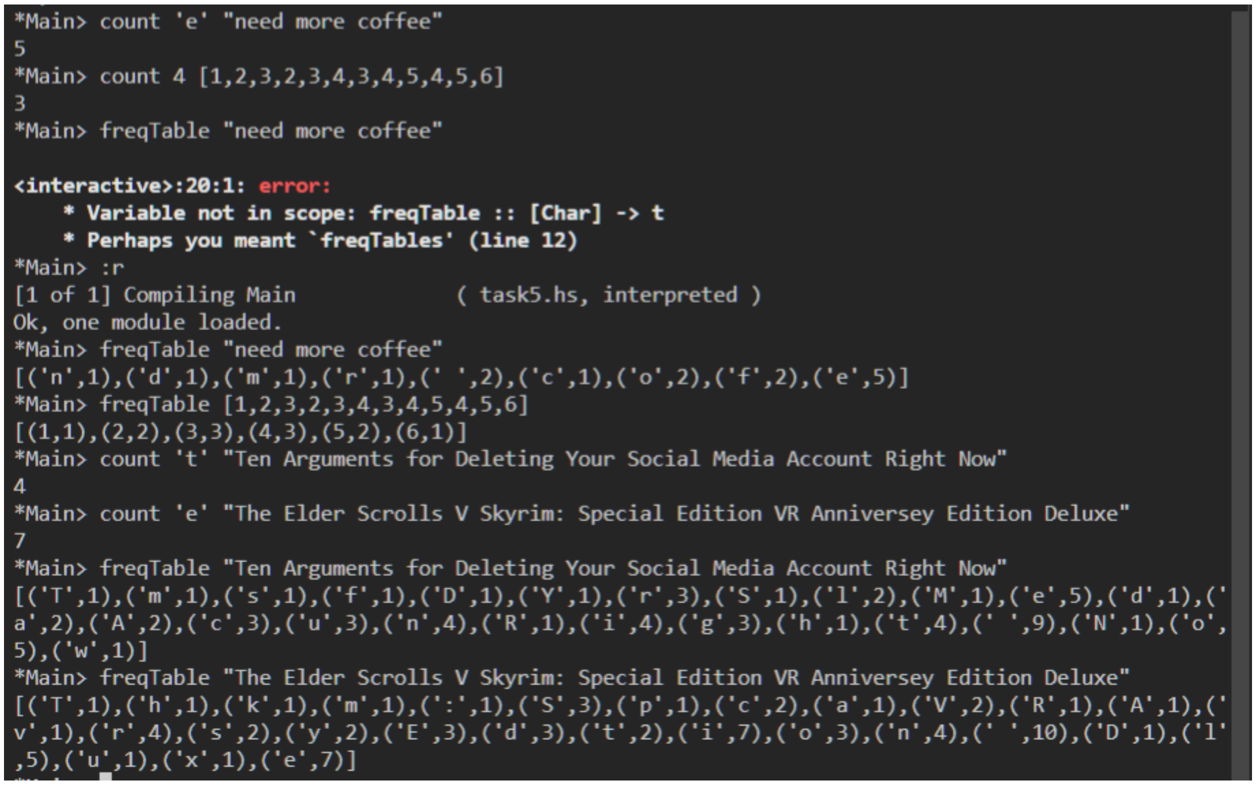


```haskell
---------------------------------
-- Part 1 - list2set ------------

list2set :: Eq a => [a] -> [a]
list2set [] = []
list2set (x:xs)
      | elem x xs = list2set(xs)
      | otherwise = x:(list2set(xs))

---------------------------------
-- Part 2 - averageWordLength ----

isPalindrome :: Eq a => [a] -> Bool
isPalindrome [] = True
isPalindrome (x:[]) = True
isPalindrome l
    | head l == last l = isPalindrome (tail(init l))
    | otherwise = False

---------------------------------
-- Part 3 - collatz -------------

collatz :: Integral a => a -> [a]
collatz n
    | n `mod` 2 == 0 = n:(collatz ( n `div` 2))
    | n == 1 = [1]
    | otherwise = n:(collatz ( 3 * n + 1))
```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

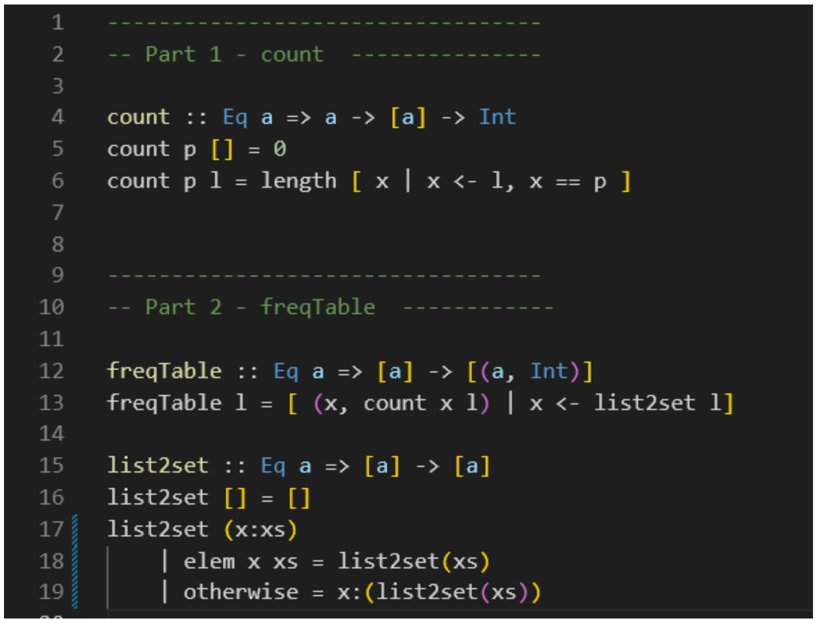
Task 5 – List Comprehensions

Demo



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

<interactive>:20:1: error:
    * Variable not in scope: freqTable :: [Char] -> t
    * Perhaps you meant `freqTables' (line 12)
*Main> :r
[1 of 1] Compiling Main             ( task5.hs, interpreted )
Ok, one module loaded.
*Main> freqTable "need more coffee"
[('n',1),('d',1),('m',1),('r',1),(' ',2),('c',1),('o',2),('f',2),('e',5)]
*Main> 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)]
*Main> count 't' "Ten Arguments for Deleting Your Social Media Account Right Now"
4
*Main> count 'e' "The Elder Scrolls V Skyrim: Special Edition VR Anniversey Edition Deluxe"
7
*Main> freqTable "Ten Arguments for Deleting Your Social Media Account Right Now"
[('T',1),('m',1),('s',1),('f',1),('D',1),('Y',1),('r',3),('S',1),('l',2),('M',1),('e',5),('d',1),('
a',2),('A',2),('c',3),('u',3),('n',4),('R',1),('i',4),('g',3),('h',1),('t',4),(' ',9),('N',1),('o',
5),('w',1)]
*Main> freqTable "The Elder Scrolls V Skyrim: Special Edition VR Anniversey Edition Deluxe"
[('T',1),('h',1),('k',1),('m',1),(':',1),('S',3),('p',1),('c',2),('a',1),('V',2),('R',1),('A',1),('
v',1),('r',4),('s',2),('y',2),('E',3),('d',3),('t',2),('i',7),('o',3),('n',4),(' ',10),('D',1),('l'
,5),('u',1),('x',1),('e',7)]
```

Code

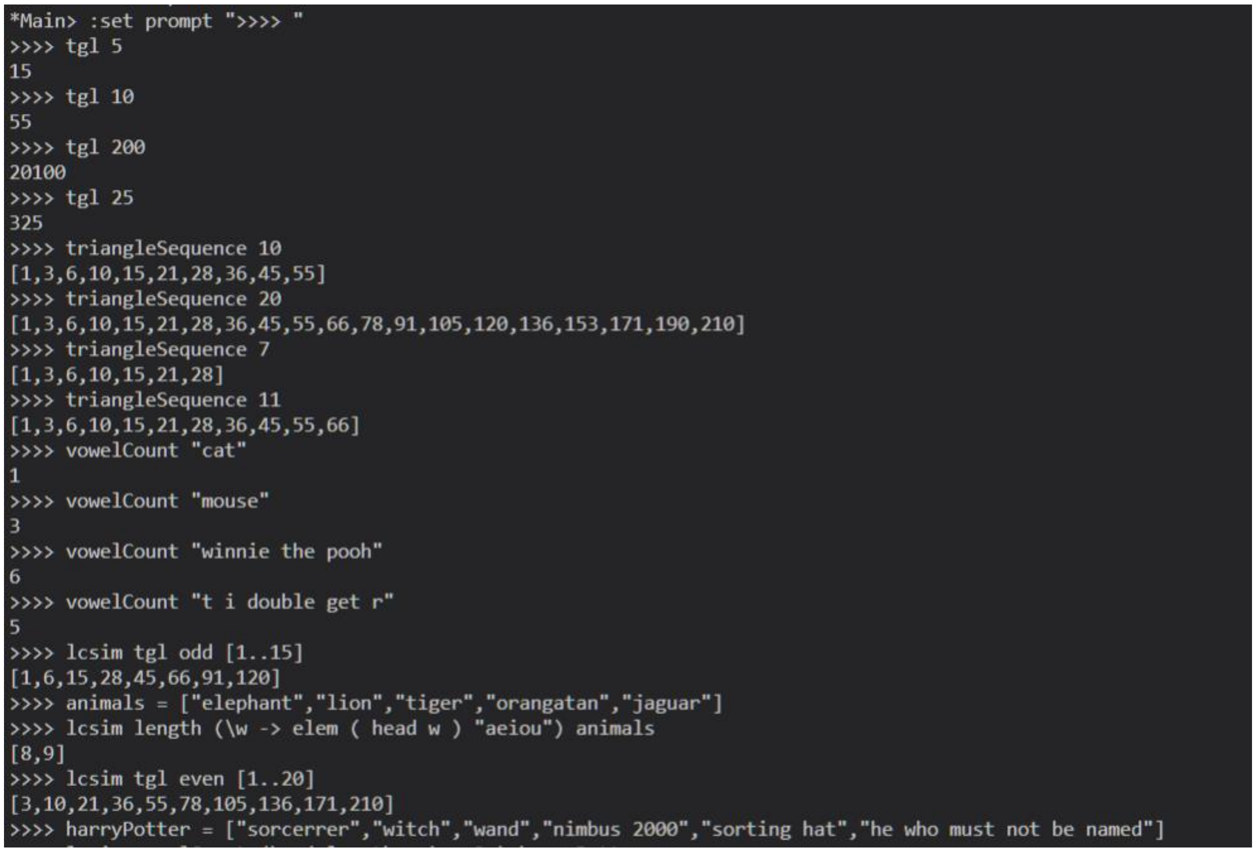


```haskell
----------------------------------
-- Part 1 - count  ---------------

count :: Eq a => a -> [a] -> Int
count p [] = 0
count p l = length [ x | x <- l, x == p ]



----------------------------------
-- Part 2 - freqTable  ------------

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

list2set :: Eq a => [a] -> [a]
list2set [] = []
list2set (x:xs)
    | elem x xs = list2set(xs)
    | otherwise = x:(list2set(xs))
```

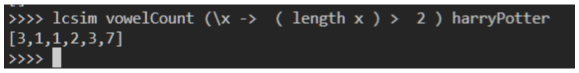
Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Task 6 – High Order Functions

Demo

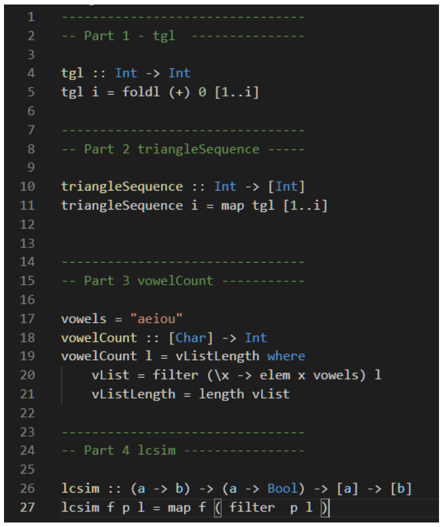


```
*Main> :set prompt ">>>> "
>>>> tgl 5
15
>>>> tgl 10
55
>>>> tgl 200
20100
>>>> tgl 25
325
>>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>>> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
>>>> triangleSequence 7
[1,3,6,10,15,21,28]
>>>> triangleSequence 11
[1,3,6,10,15,21,28,36,45,55,66]
>>>> vowelCount "cat"
1
>>>> vowelCount "mouse"
3
>>>> vowelCount "winnie the pooh"
6
>>>> vowelCount "t i double get r"
5
>>>> 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]
>>>> lcsim tgl even [1..20]
[3,10,21,36,55,78,105,136,171,210]
>>>> harryPotter = ["sorcerrer","witch","wand","nimbus 2000","sorting hat","he who must not be named"]
```



```
>>>> lcsim vowelCount (\x ->  ( length x ) >  2 ) harryPotter
[3,1,1,2,3,7]
>>>>
```

Demo



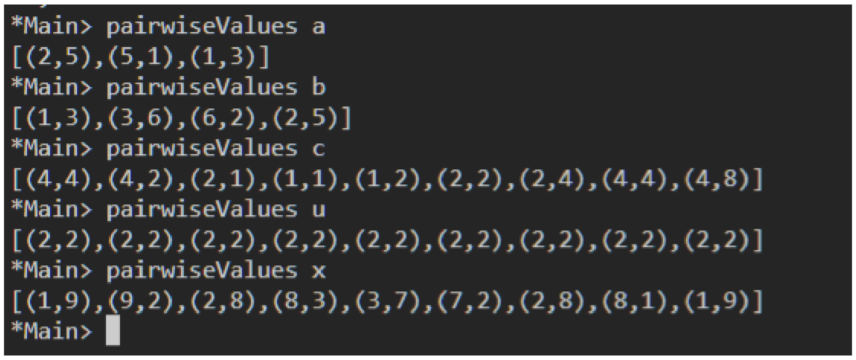
```haskell
--------------------------------
-- Part 1 - tgl  --------------

tgl :: Int -> Int
tgl i = foldl (+) 0 [1..i]

--------------------------------
-- Part 2 triangleSequence -----

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


--------------------------------
-- Part 3 vowelCount ----------

vowels = "aeiou"
vowelCount :: [Char] -> Int
vowelCount l = vListLength where
    vList = filter (\x -> elem x vowels) l
    vListLength = length vList

--------------------------------
-- Part 4 lcsim ---------------

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

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Task 7 – An Interesting Statistic: nPVI

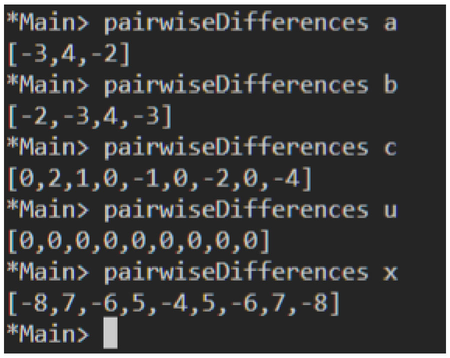
Demos

pairwiseValues



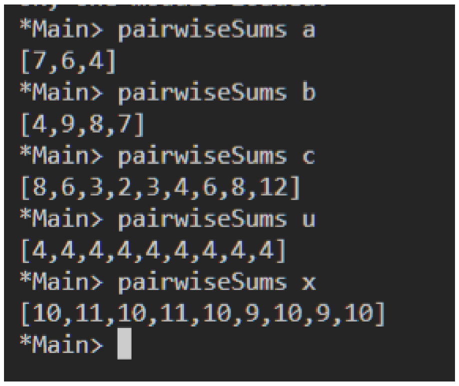
```
*Main> pairwiseValues a
[(2,5),(5,1),(1,3)]
*Main> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
*Main> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
*Main> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
*Main> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]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

pairwiseDifferences



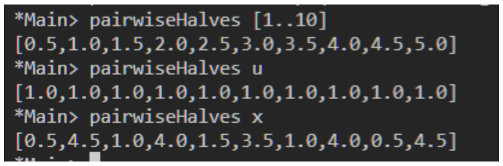
```
*Main> pairwiseDifferences a
[-3,4,-2]
*Main> pairwiseDifferences b
[-2,-3,4,-3]
*Main> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
*Main> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
*Main> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
*Main>
```

pairwiseSums



```
*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]
*Main>
```

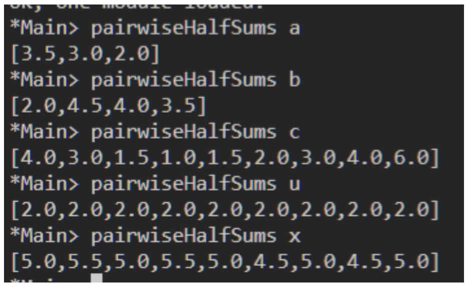
pairwiseHalves



```
*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]
*Main> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

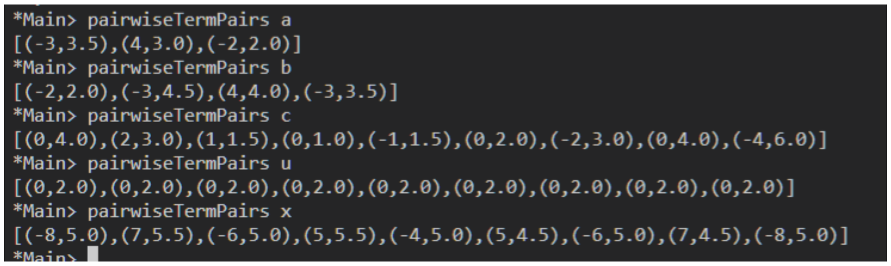
Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

pairwiseHalfSums



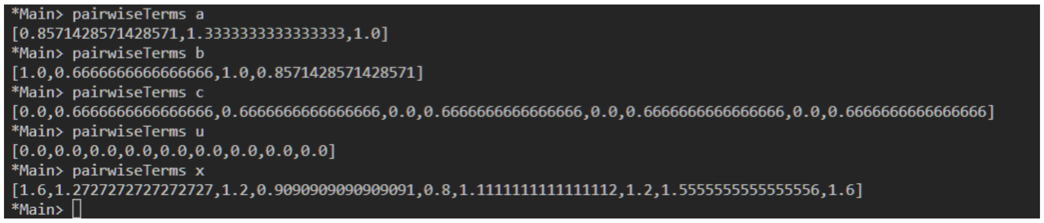
```
*Main> pairwiseHalfSums a
[3.5,3.0,2.0]
*Main> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
*Main> pairwiseHalfSums c
[4.0,3.0,1.5,1.0,1.5,2.0,3.0,4.0,6.0]
*Main> pairwiseHalfSums u
[2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0,2.0]
*Main> pairwiseHalfSums x
[5.0,5.5,5.0,5.5,5.0,4.5,5.0,4.5,5.0]
```

pairwiseTermPairs



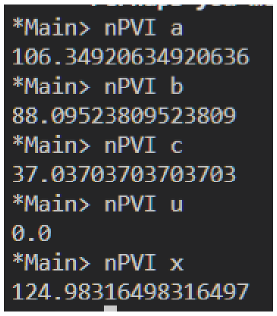
```
*Main> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
*Main> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
*Main> pairwiseTermPairs c
[(0,4.0),(2,3.0),(1,1.5),(0,1.0),(-1,1.5),(0,2.0),(-2,3.0),(0,4.0),(-4,6.0)]
*Main> pairwiseTermPairs u
[(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0),(0,2.0)]
*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)]
```

pairwiseTerms



```
*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]
*Main>
```

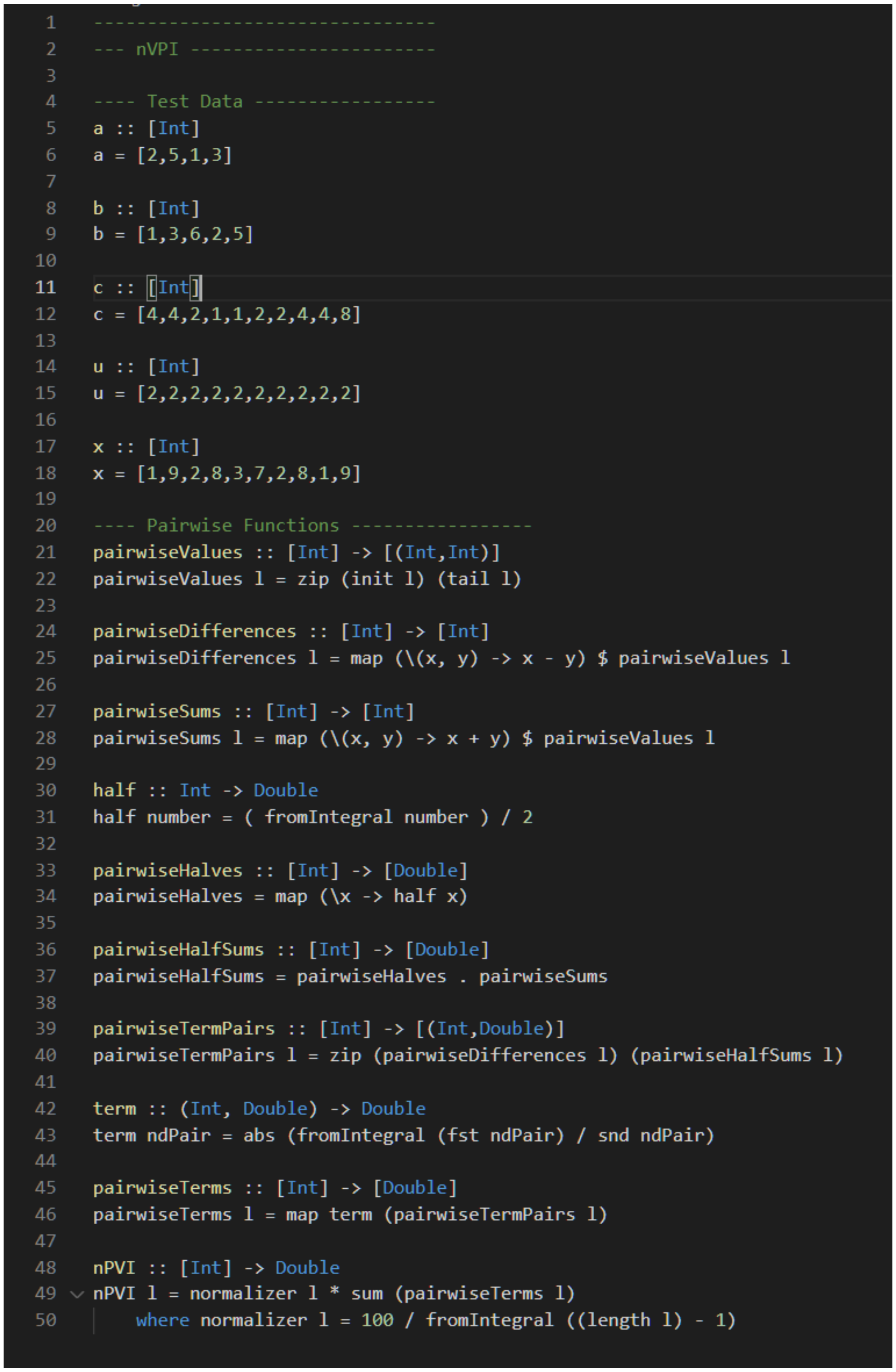
nPVI



```
*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
```

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Demos

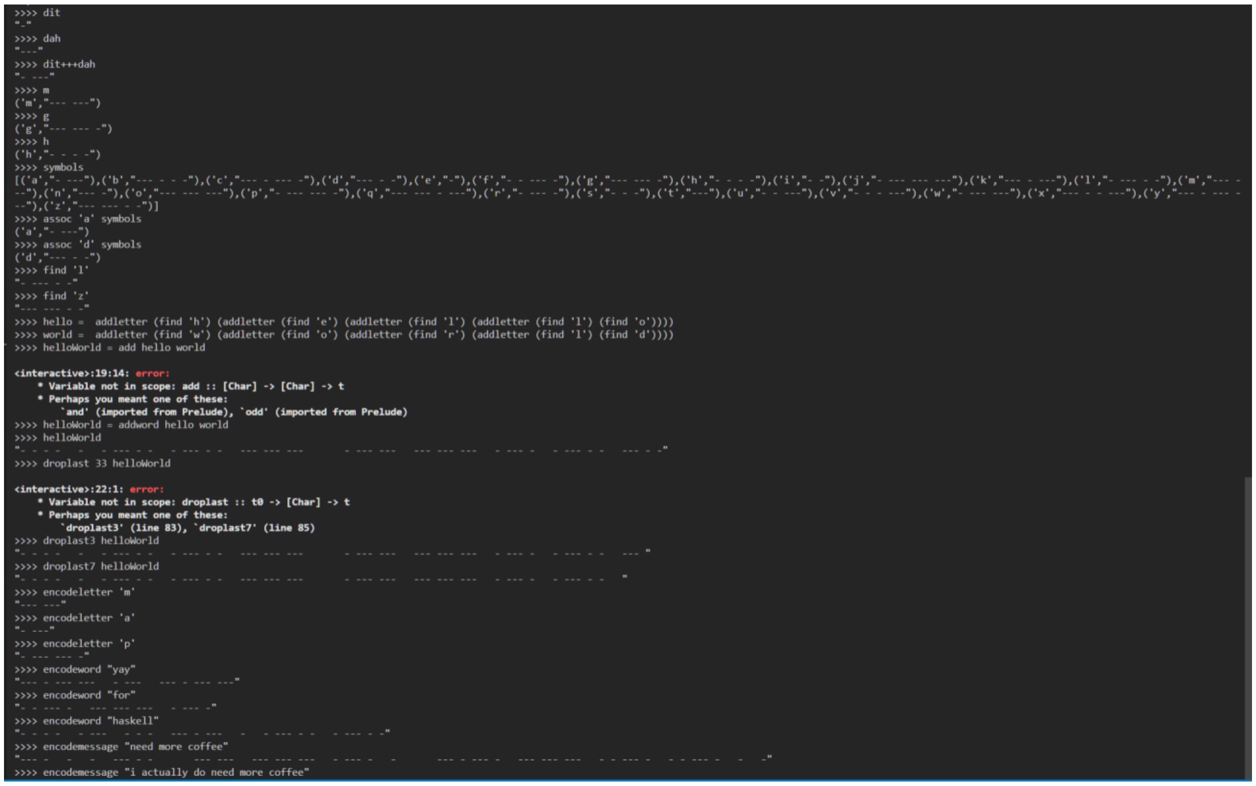


```haskell
--------------------------------
--- nVPI ----------------------

---- 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]

---- Pairwise Functions -----------------
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues l = zip (init l) (tail l)

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

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

half :: Int -> Double
half number = ( fromIntegral number ) / 2

pairwiseHalves :: [Int] -> [Double]
pairwiseHalves = map (\x -> half x)

pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums = pairwiseHalves . pairwiseSums

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

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

pairwiseTerms :: [Int] -> [Double]
pairwiseTerms l = map term (pairwiseTermPairs l)

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

Nathaniel Wolf
Haskell Programming Assignment: Various Computation
5/1/2022
CSC 344

Task 8 – Historic Code: The Dit Dah Code

Subtask 8a, 8b, 8c, 8d



```
>>>> dit
"-"
>>>> dah
"---"
>>>> 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',"--- --- - -")]
>>>> assoc 'a' symbols
('a',"- ---")
>>>> assoc 'd' symbols
('d',"--- - -")
>>>> find 'l'
"- --- - -"
>>>> find 'z'
"--- --- - -"
>>>> hello =  addletter (find 'h') (addletter (find 'e') (addletter (find 'l') (addletter (find 'l') (find 'o'))))
>>>> world =  addletter (find 'w') (addletter (find 'o') (addletter (find 'r') (addletter (find 'l') (find 'd'))))
>>>> helloWorld = add hello world

<interactive>:19:14: error:
    * Variable not in scope: add :: [Char] -> [Char] -> t
    * Perhaps you meant one of these:
        `and' (imported from Prelude), `odd' (imported from Prelude)
>>>> helloWorld = addword hello world
>>>> helloWorld
"- - - -   -   - --- - -   - --- - -   --- --- ---       - --- ---   --- --- ---   - --- -   - --- - -   --- - -"
>>>> droplast 33 helloWorld

<interactive>:22:1: error:
    * Variable not in scope: droplast :: t0 -> [Char] -> t
    * Perhaps you meant one of these:
        `droplast3' (line 83), `droplast7' (line 85)
>>>> droplast3 helloWorld
"- - - -   -   - --- - -   - --- - -   --- --- ---       - --- ---   --- --- ---   - --- -   - --- - -   --- "
>>>> droplast7 helloWorld
"- - - -   -   - --- - -   - --- - -   --- --- ---       - --- ---   --- --- ---   - --- -   - --- - -   "
>>>> encodeletter 'm'
"--- ---"
>>>> encodeletter 'a'
"- ---"
>>>> encodeletter 'p'
"- --- --- -"
>>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>>> encodeword "for"
"- - --- -   --- --- ---   - --- -"
>>>> encodeword "haskell"
"- - - -   - ---   - - -   --- - ---   -   - --- - -   - --- - -"
>>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
>>>> encodemessage "i actually do need more coffee"
```