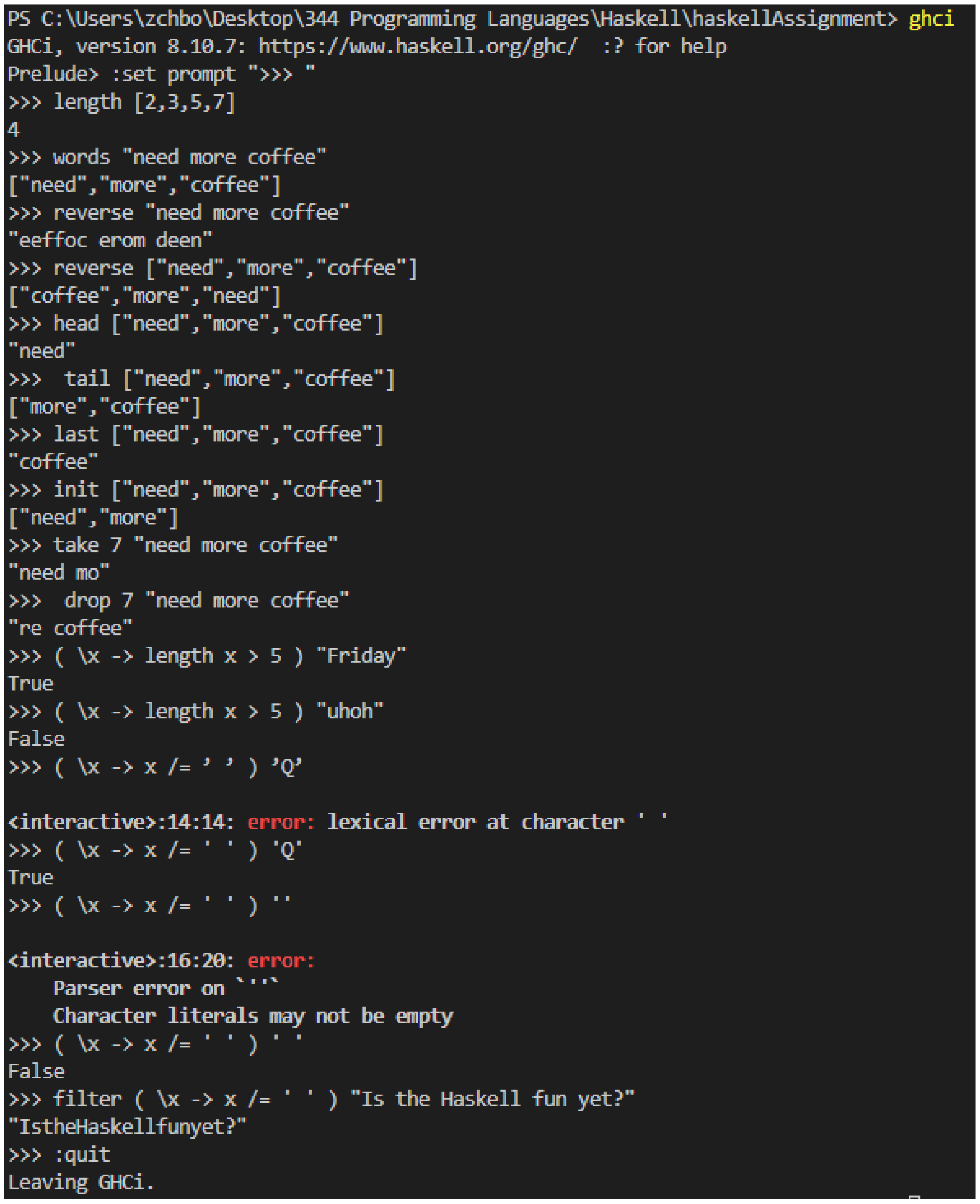
Haskell Programming Assignment: Various Computations

Learning Abstract

This Haskell Programming Assignment: Various Computations is a total of 8 tasks. Task 1 involves mimicking a demo to gain familiarity with Haskell and list manipulation. Task 2-6 focus on high order functions, recursive functions, and more complex tasks. Task 7 we create an nVPI and task 8 we use a Haskell program provided to perform the different functions included and see how they work.

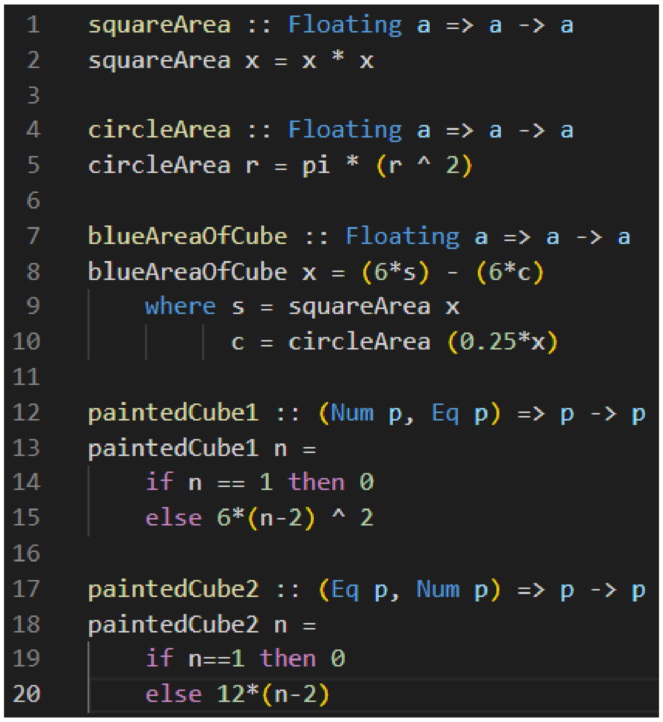
Task 1: Mindfully Mimicking the Demo



```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\haskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
>>> length [2,3,5,7]
4
>>> words "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>:14:14: error: lexical error at character ' '
>>> ( \x -> x /= ' ' ) 'Q'
True
>>> ( \x -> x /= ' ' ) ''

<interactive>:16:20: error:
    Parser error on `''`
    Character literals may not be empty
>>> ( \x -> x /= ' ' ) ' '
False
>>> filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>> :quit
Leaving GHCi.
```

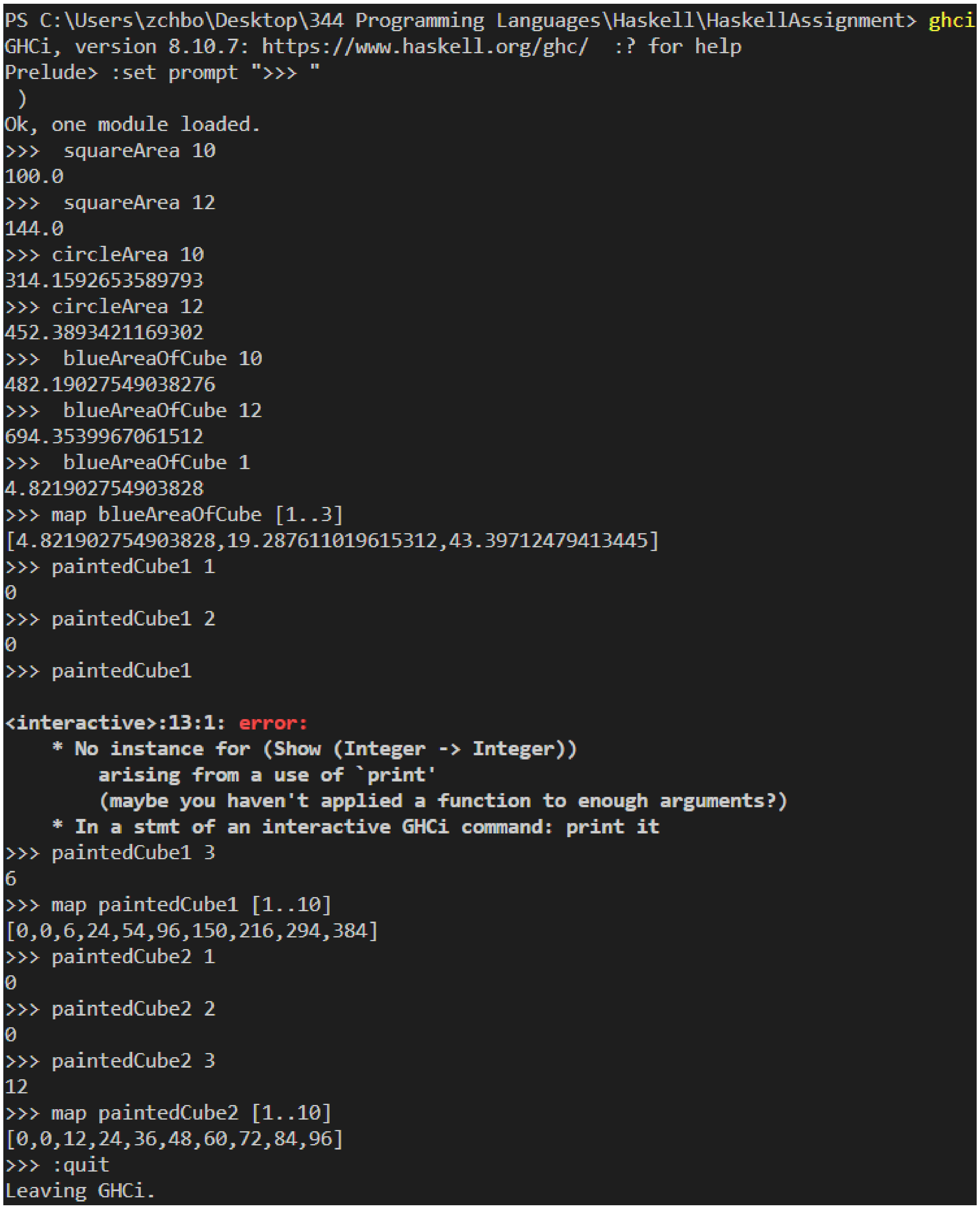
Task 2: Numeric Function Definitions

Code:

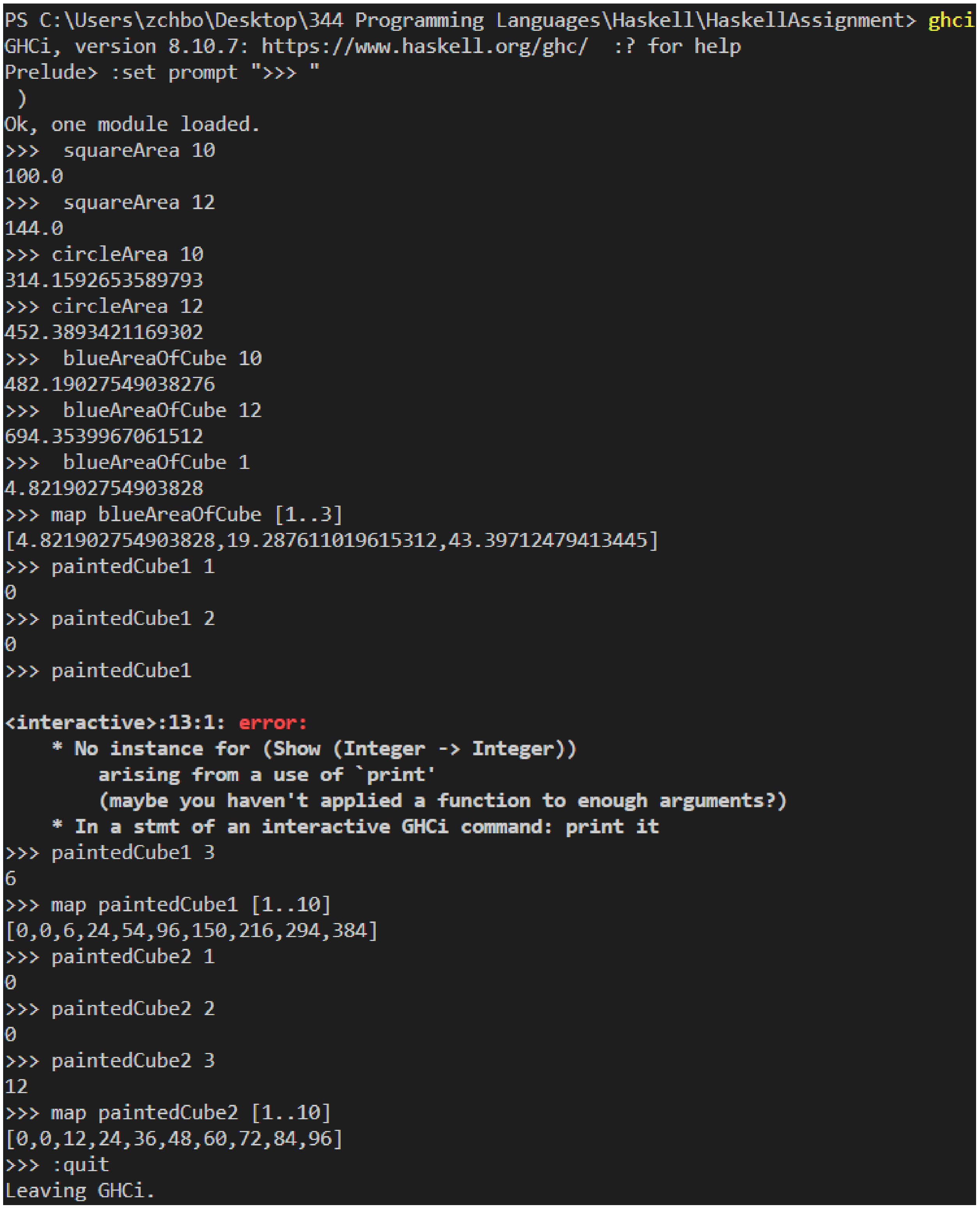


```haskell
squareArea :: Floating a => a -> a
squareArea x = x * x

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

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

paintedCube1 :: (Num p, Eq p) => p -> p
paintedCube1 n =
    if n == 1 then 0
    else 6*(n-2) ^ 2

paintedCube2 :: (Eq p, Num p) => p -> p
paintedCube2 n =
    if n==1 then 0
    else 12*(n-2)
```

Demo:



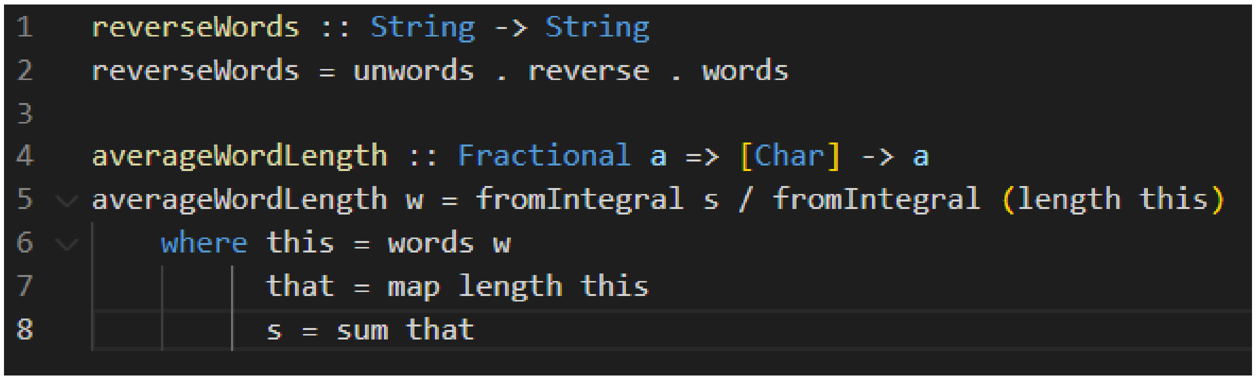
```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
 )
Ok, one module loaded.
>>>  squareArea 10
100.0
>>>  squareArea 12
144.0
>>> circleArea 10
314.1592653589793
>>> circleArea 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

<interactive>:13:1: error:
    * No instance for (Show (Integer -> Integer))
        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
>>> 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.
```



```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
 )
Ok, one module loaded.
>>>  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

<interactive>:13:1: error:
    * No instance for (Show (Integer -> Integer))
        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
>>> 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.
```

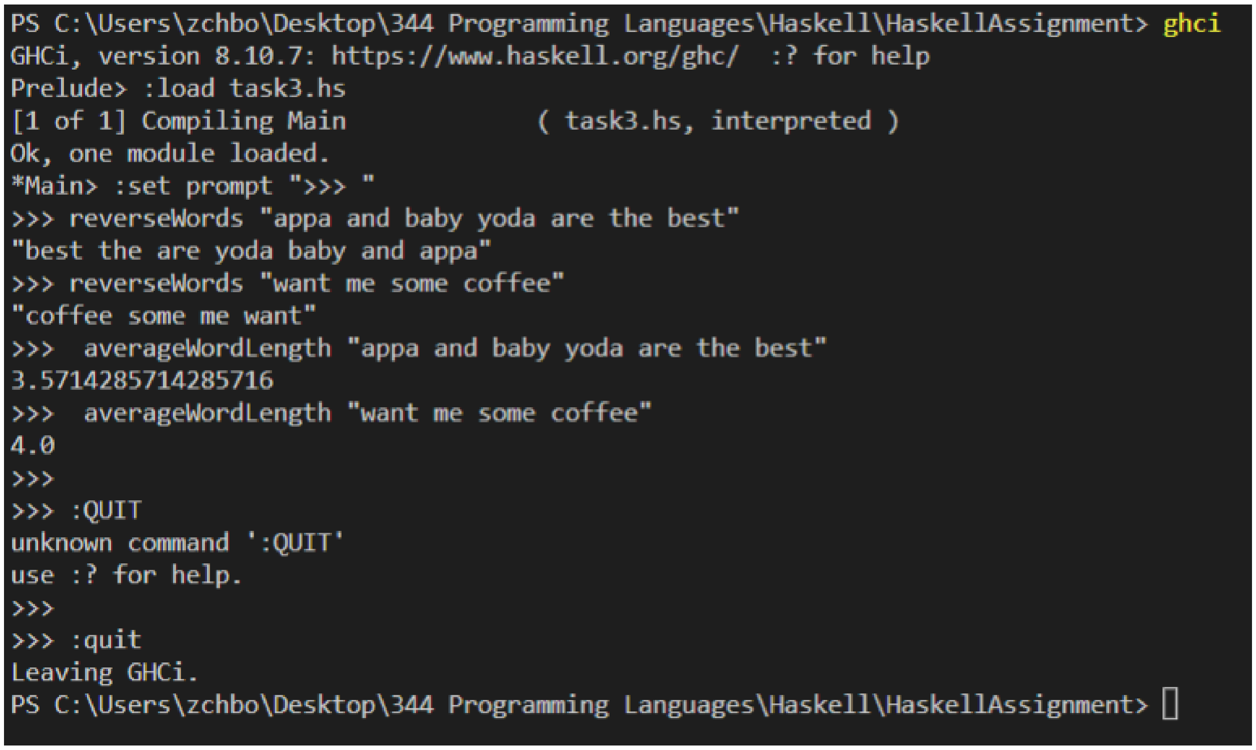
Task 3: Puzzlers

Code:



```haskell
reverseWords :: String -> String
reverseWords = unwords . reverse . words

averageWordLength :: Fractional a => [Char] -> a
averageWordLength w = fromIntegral s / fromIntegral (length this)
    where this = words w
          that = map length this
          s = sum that
```

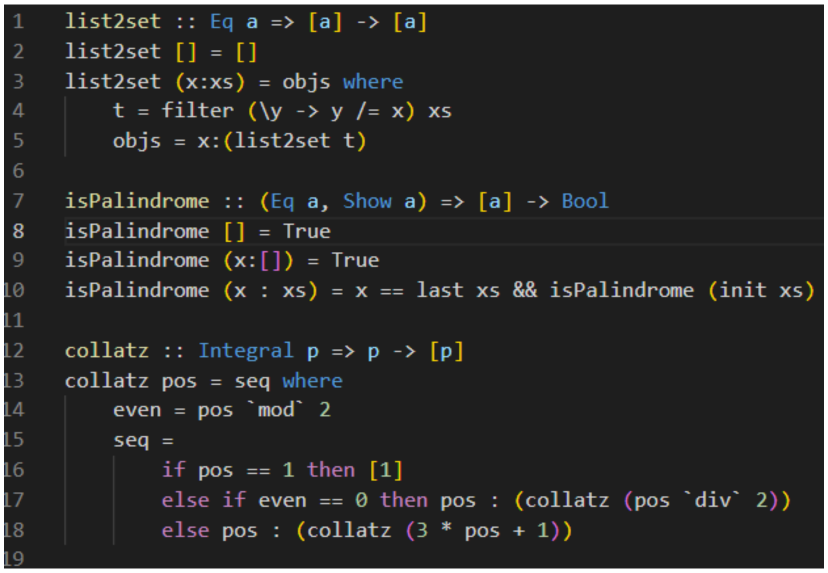
Demo:



```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :load task3.hs
[1 of 1] Compiling Main             ( task3.hs, interpreted )
Ok, one module loaded.
*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
>>>
>>> :QUIT
unknown command ':QUIT'
use :? for help.
>>>
>>> :quit
Leaving GHCi.
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment>
```

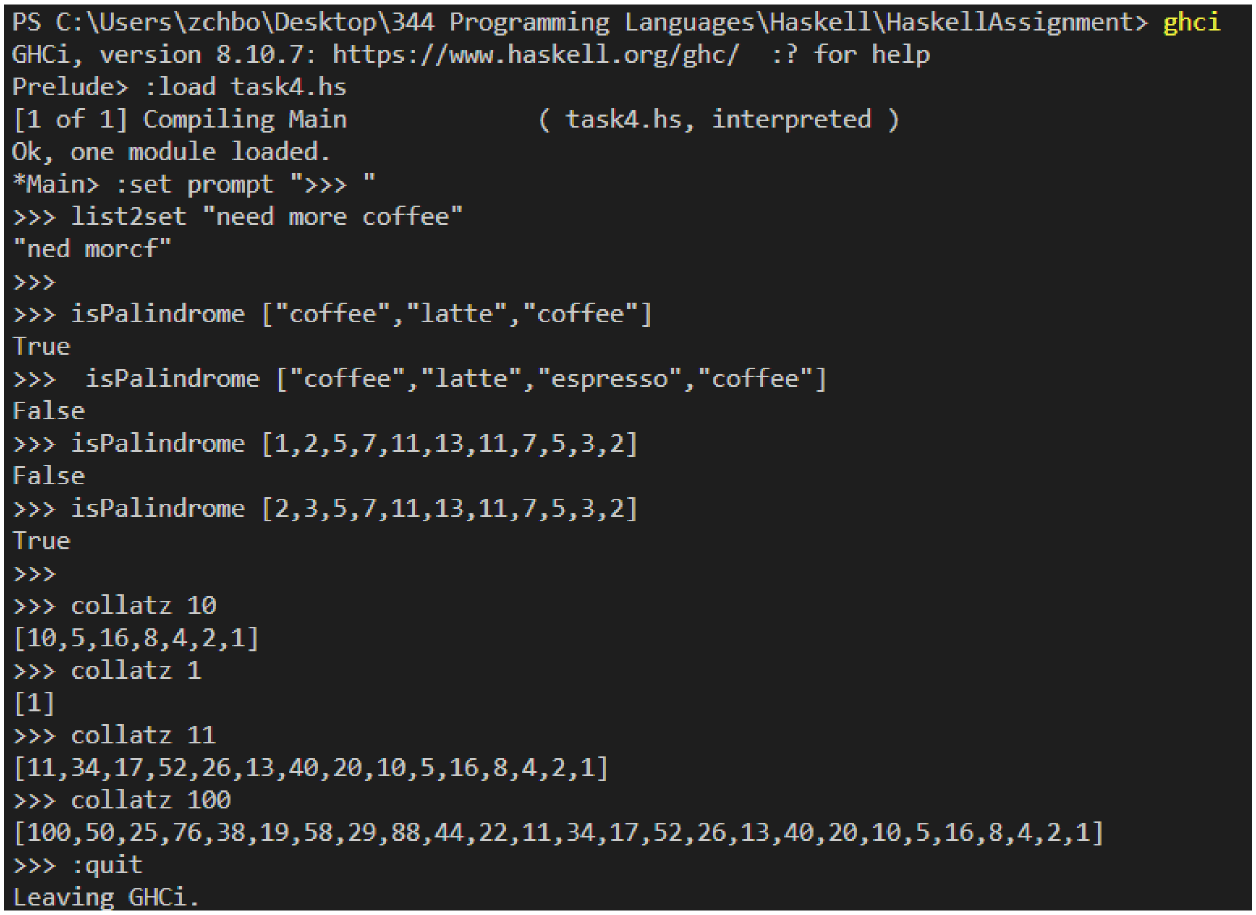
Task 4: Recursive List Processors

Code:



```haskell
list2set :: Eq a => [a] -> [a]
list2set [] = []
list2set (x:xs) = objs where
    t = filter (\y -> y /= x) xs
    objs = x:(list2set t)

isPalindrome :: (Eq a, Show a) => [a] -> Bool
isPalindrome [] = True
isPalindrome (x:[]) = True
isPalindrome (x : xs) = x == last xs && isPalindrome (init xs)

collatz :: Integral p => p -> [p]
collatz pos = seq where
    even = pos `mod` 2
    seq =
        if pos == 1 then [1]
        else if even == 0 then pos : (collatz (pos `div` 2))
        else pos : (collatz (3 * pos + 1))
```

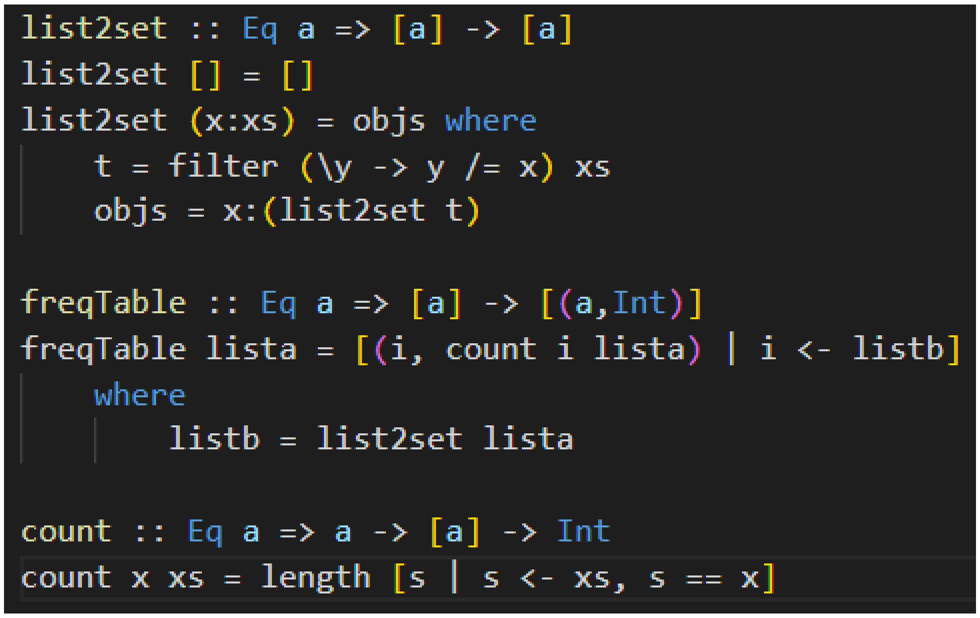
Demo:



```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :load task4.hs
[1 of 1] Compiling Main             ( task4.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> list2set "need more coffee"
"ned morcf"
>>>
>>> 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 1
[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.
```

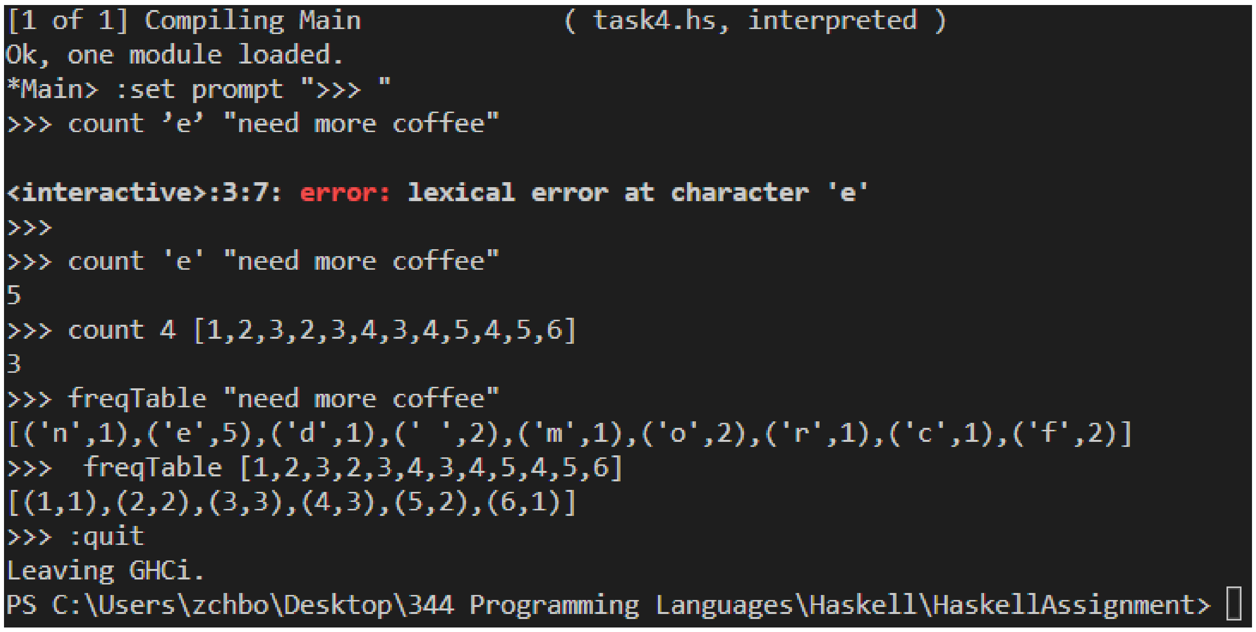
Task 5: List Comprehensions

Code:



```haskell
list2set :: Eq a => [a] -> [a]
list2set [] = []
list2set (x:xs) = objs where
    t = filter (\y -> y /= x) xs
    objs = x:(list2set t)

freqTable :: Eq a => [a] -> [(a,Int)]
freqTable lista = [(i, count i lista) | i <- listb]
    where
        listb = list2set lista

count :: Eq a => a -> [a] -> Int
count x xs = length [s | s <- xs, s == x]
```

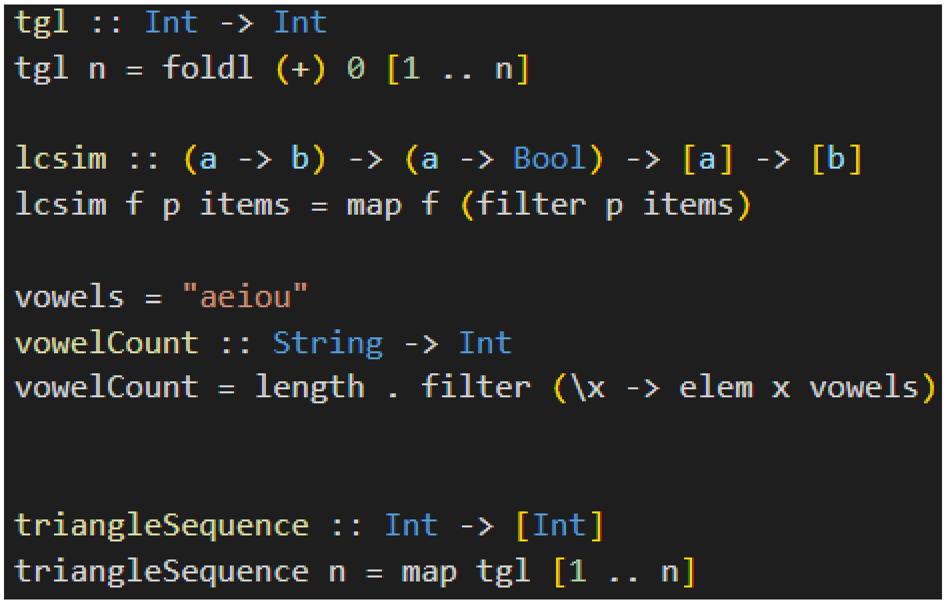
Demo:



```
[1 of 1] Compiling Main             ( task4.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> count ’e’ "need more coffee"

<interactive>:3:7: error: lexical error at character 'e'
>>>
>>> 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),('e',5),('d',1),(' ',2),('m',1),('o',2),('r',1),('c',1),('f',2)]
>>>  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.
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment>
```

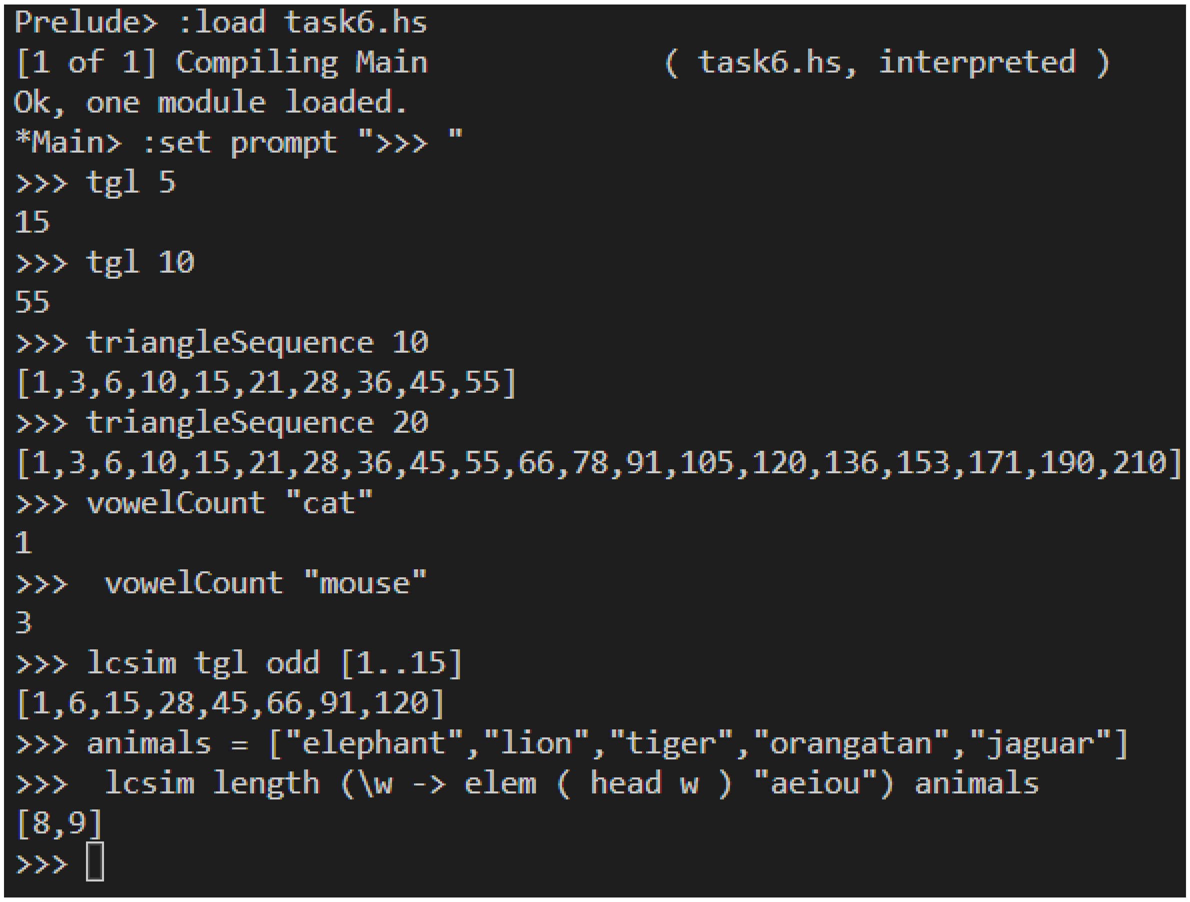
Task 6: Higher Order Functions

Code:



```haskell
tgl :: Int -> Int
tgl n = foldl (+) 0 [1 .. n]

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

vowels = "aeiou"
vowelCount :: String -> Int
vowelCount = length . filter (\x -> elem x vowels)


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

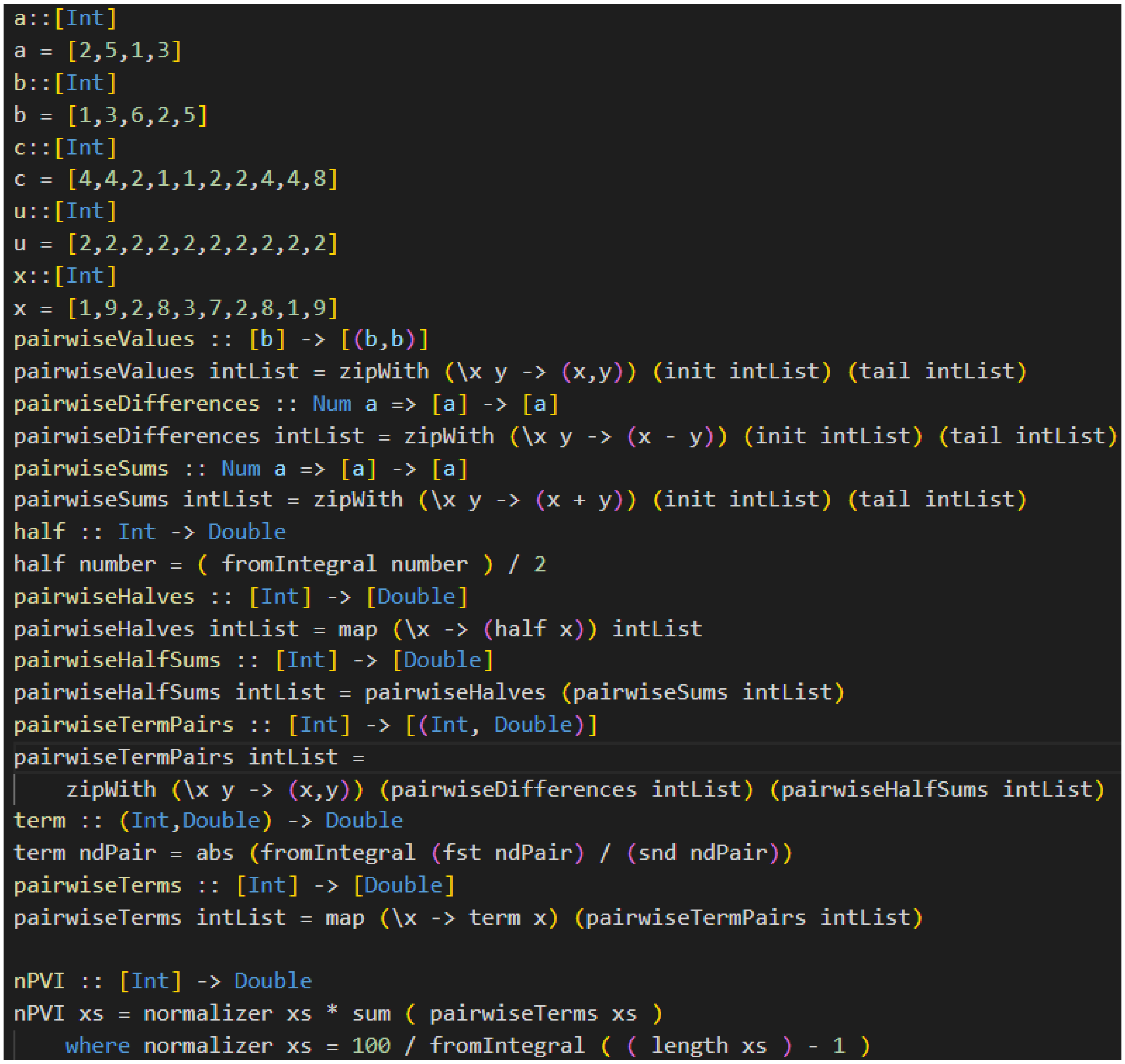
Demo:



```
Prelude> :load task6.hs
[1 of 1] Compiling Main             ( task6.hs, interpreted )
Ok, one module loaded.
*Main> :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]
>>> 
```

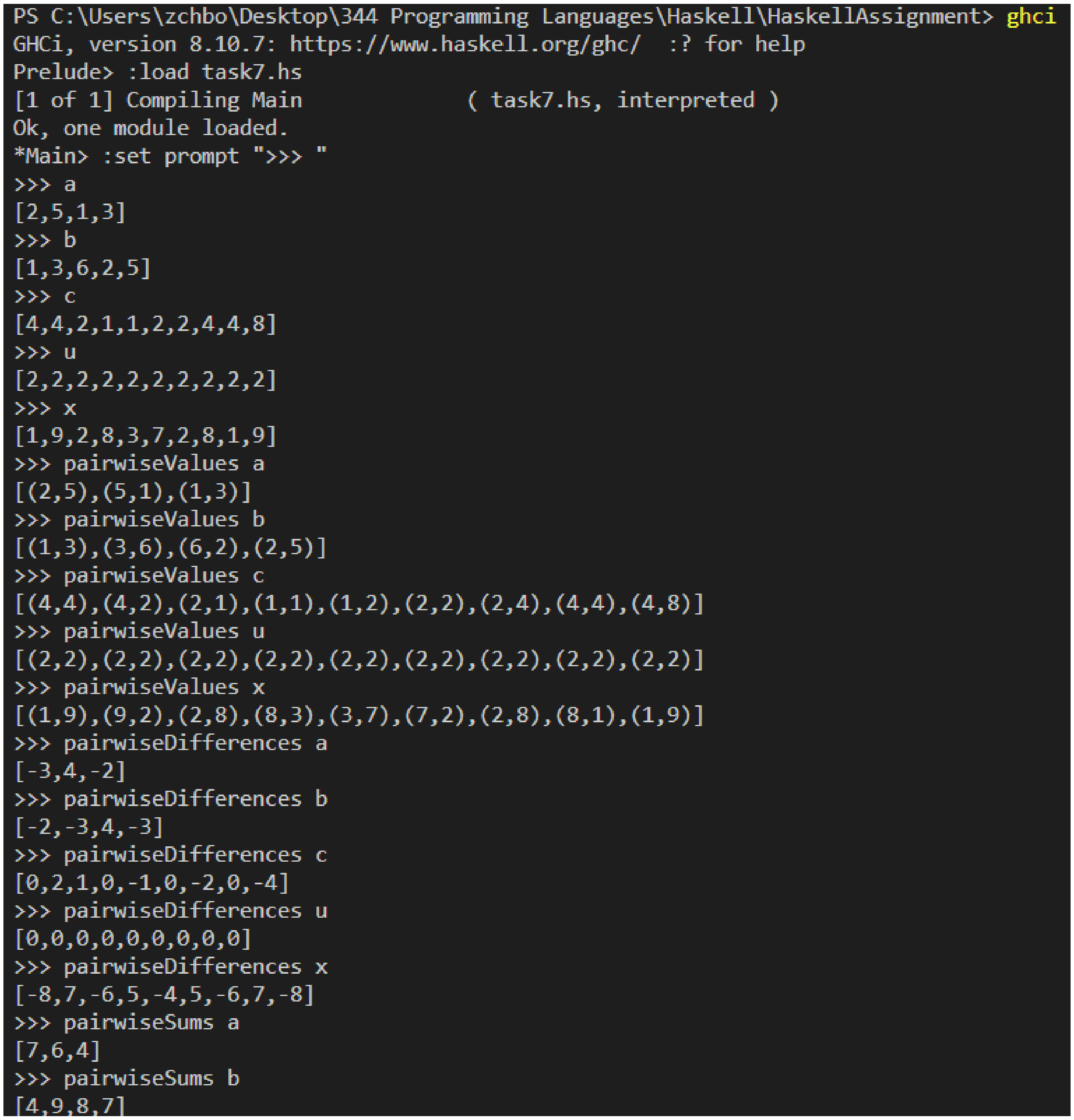
Task 7: An Interesting Statistic: nPVI

Code:

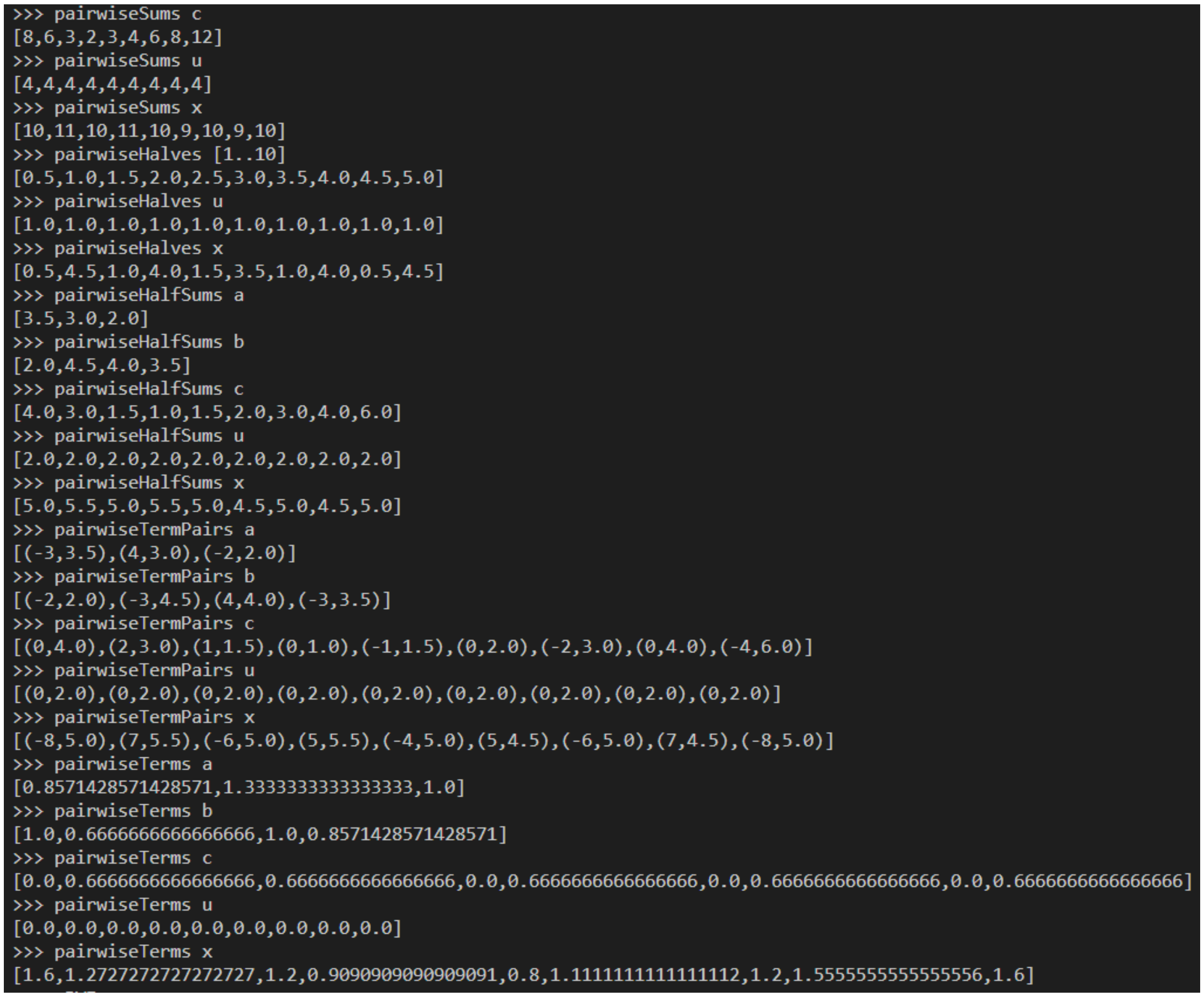


```haskell
a::[Int]
a = [2,5,1,3]
b::[Int]
b = [1,3,6,2,5]
c::[Int]
c = [4,4,2,1,1,2,2,4,4,8]
u::[Int]
u = [2,2,2,2,2,2,2,2,2,2]
x::[Int]
x = [1,9,2,8,3,7,2,8,1,9]
pairwiseValues :: [b] -> [(b,b)]
pairwiseValues intList = zipWith (\x y -> (x,y)) (init intList) (tail intList)
pairwiseDifferences :: Num a => [a] -> [a]
pairwiseDifferences intList = zipWith (\x y -> (x - y)) (init intList) (tail intList)
pairwiseSums :: Num a => [a] -> [a]
pairwiseSums intList = zipWith (\x y -> (x + y)) (init intList) (tail intList)
half :: Int -> Double
half number = ( fromIntegral number ) / 2
pairwiseHalves :: [Int] -> [Double]
pairwiseHalves intList = map (\x -> (half x)) intList
pairwiseHalfSums :: [Int] -> [Double]
pairwiseHalfSums intList = pairwiseHalves (pairwiseSums intList)
pairwiseTermPairs :: [Int] -> [(Int, Double)]
pairwiseTermPairs intList =
    zipWith (\x y -> (x,y)) (pairwiseDifferences intList) (pairwiseHalfSums intList)
term :: (Int,Double) -> Double
term ndPair = abs (fromIntegral (fst ndPair) / (snd ndPair))
pairwiseTerms :: [Int] -> [Double]
pairwiseTerms intList = map (\x -> term x) (pairwiseTermPairs intList)

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

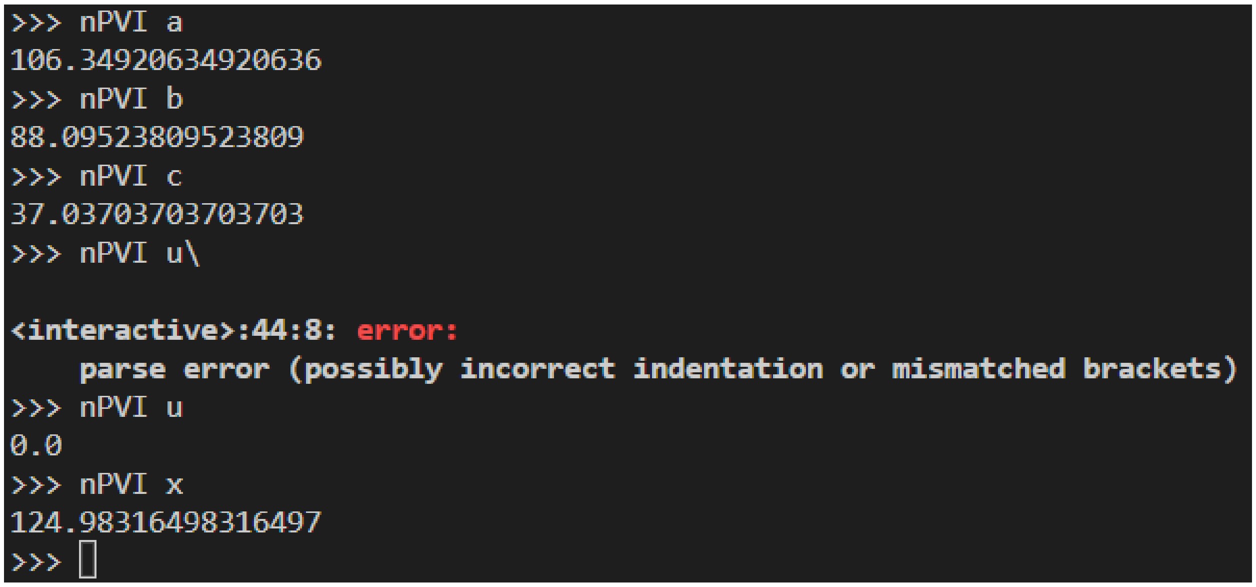
Demos:



```
PS C:\Users\zchbo\Desktop\344 Programming Languages\Haskell\HaskellAssignment> ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :load task7.hs
[1 of 1] Compiling Main             ( task7.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> 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]
>>> 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)]
>>> 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]
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
```



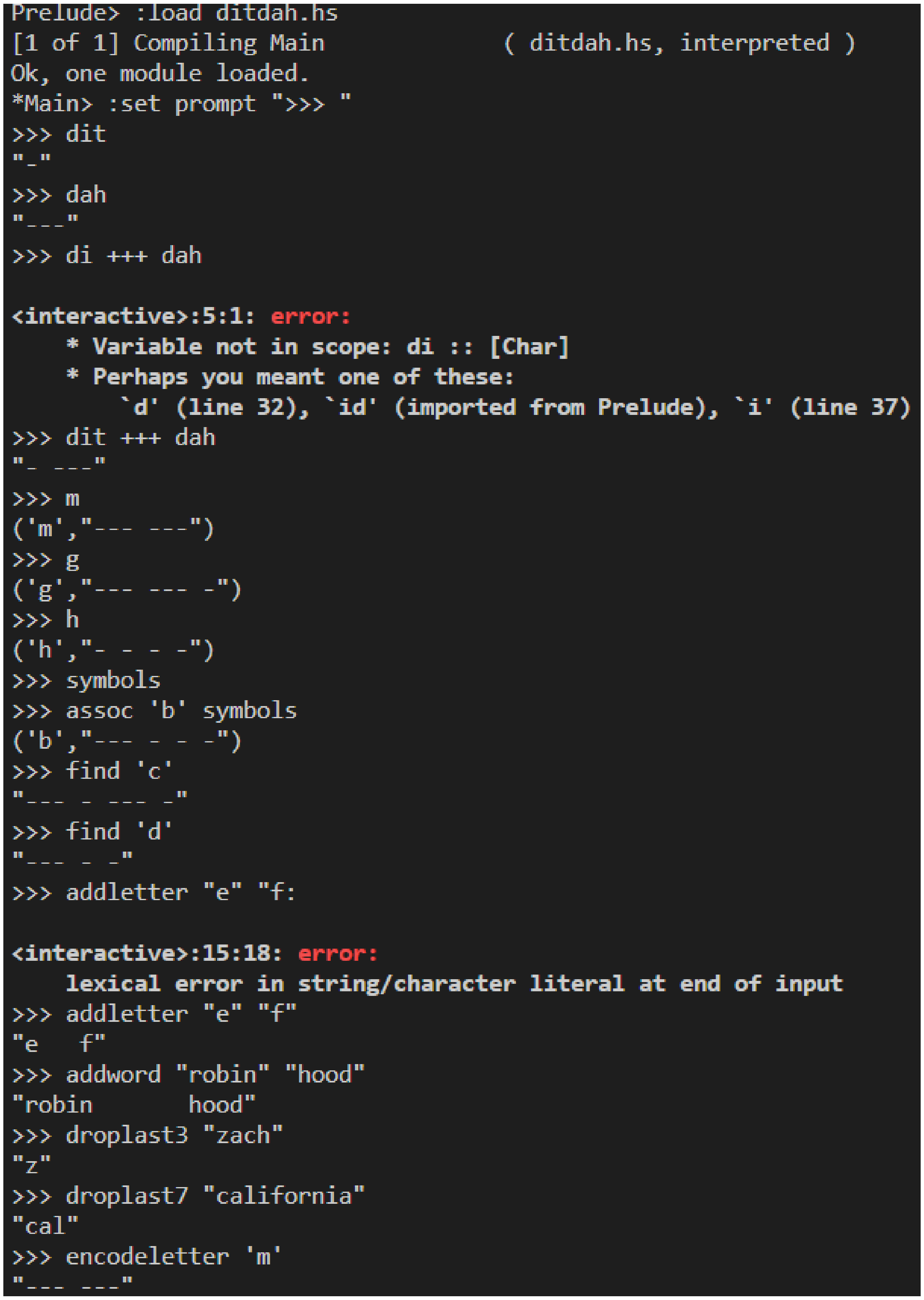
```
>>> pairwiseSums c
[8,6,3,2,3,4,6,8,12]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
>>> pairwiseHalves [1..10]
[0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
>>> pairwiseHalves u
[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
>>> pairwiseHalves x
[0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
>>> 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]
>>> 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)]
>>> 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]
```



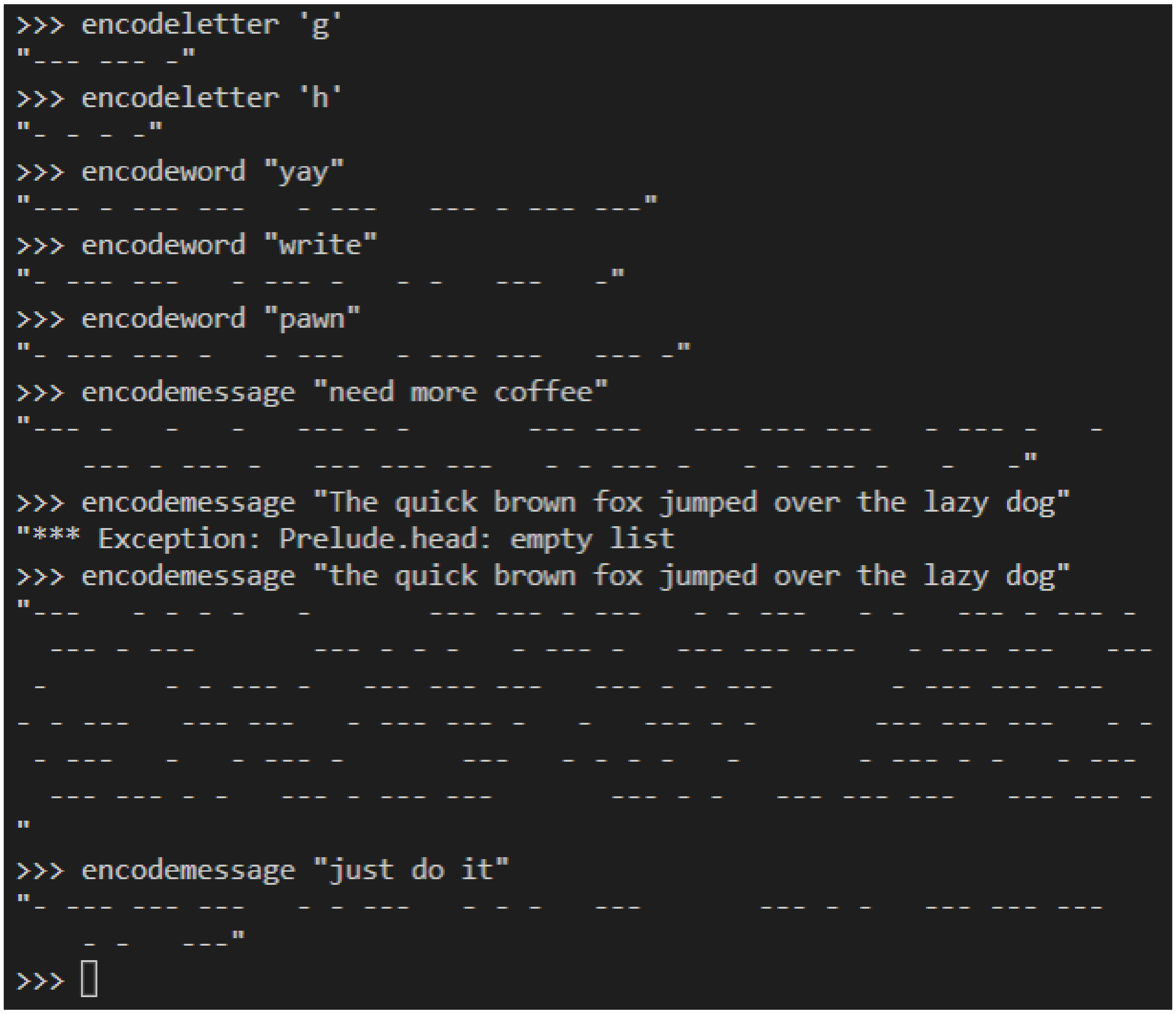
```
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u\

<interactive>:44:8: error:
    parse error (possibly incorrect indentation or mismatched brackets)
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
>>> 
```

Task 8: Historic Code: The Dit Dah Code

Demos:



```
Prelude> :load ditdah.hs
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> dit
"-"
>>> dah
"---"
>>> di +++ dah

<interactive>:5:1: error:
    * Variable not in scope: di :: [Char]
    * Perhaps you meant one of these:
        `d' (line 32), `id' (imported from Prelude), `i' (line 37)
>>> dit +++ dah
"- ---"
>>> m
('m',"--- ---")
>>> g
('g',"--- --- -")
>>> h
('h',"- - - -")
>>> symbols
>>> assoc 'b' symbols
('b',"--- - - -")
>>> find 'c'
"--- - --- -"
>>> find 'd'
"--- - -"
>>> addletter "e" "f:

<interactive>:15:18: error:
    lexical error in string/character literal at end of input
>>> addletter "e" "f"
"e   f"
>>> addword "robin" "hood"
"robin       hood"
>>> droplast3 "zach"
"z"
>>> droplast7 "california"
"cal"
>>> encodeletter 'm'
"--- ---"
```



```
>>> encodeletter 'g'
"--- --- -"
>>> encodeletter 'h'
"- - - -"
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "write"
"- --- ---   - --- -   - -   ---   -"
>>> encodeword "pawn"
"- --- --- -   - ---   - --- ---   --- -"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -
    --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
>>> encodemessage "The quick brown fox jumped over the lazy dog"
"*** Exception: Prelude.head: empty list
>>> encodemessage "the quick brown fox jumped over the lazy dog"
"---   - - - -   -       --- --- - ---   - - ---   - -   --- - --- -
  --- - ---       --- - - -   - --- -   --- --- ---   - --- ---   ---
 -       - - --- -   --- --- ---   --- - - ---       - --- --- ---
- - ---   --- ---   - --- --- -   -   --- - -       --- --- ---   - -
 - ---   -   - --- -       ---   - - - -   -       - --- - -   - ---
  --- --- - -   --- - --- ---       --- - -   --- --- ---   --- --- -
"
>>> encodemessage "just do it"
"- --- --- ---   - - ---   - - -   ---       --- - -   --- --- ---
    - -   ---"
>>> 
```