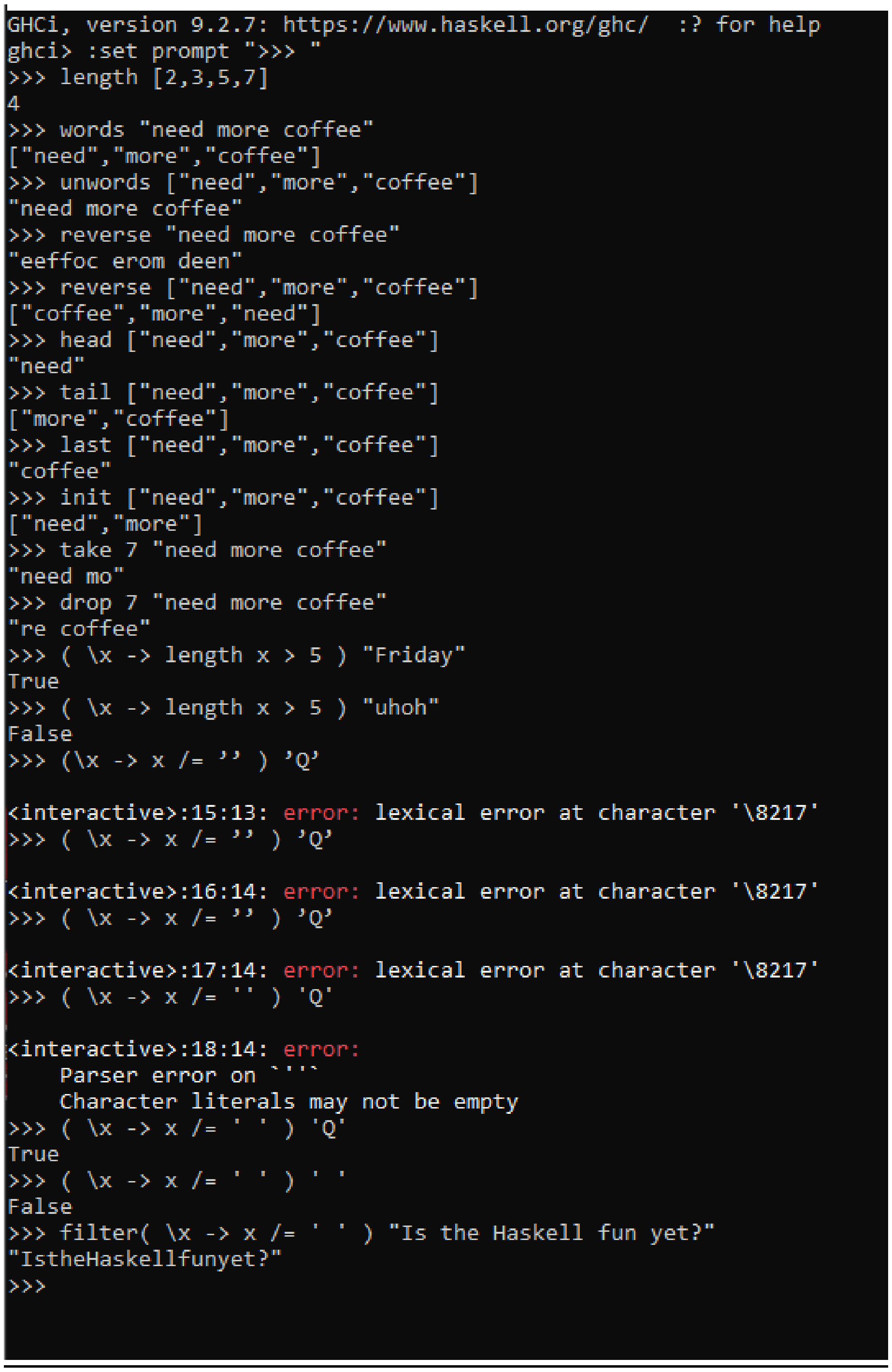
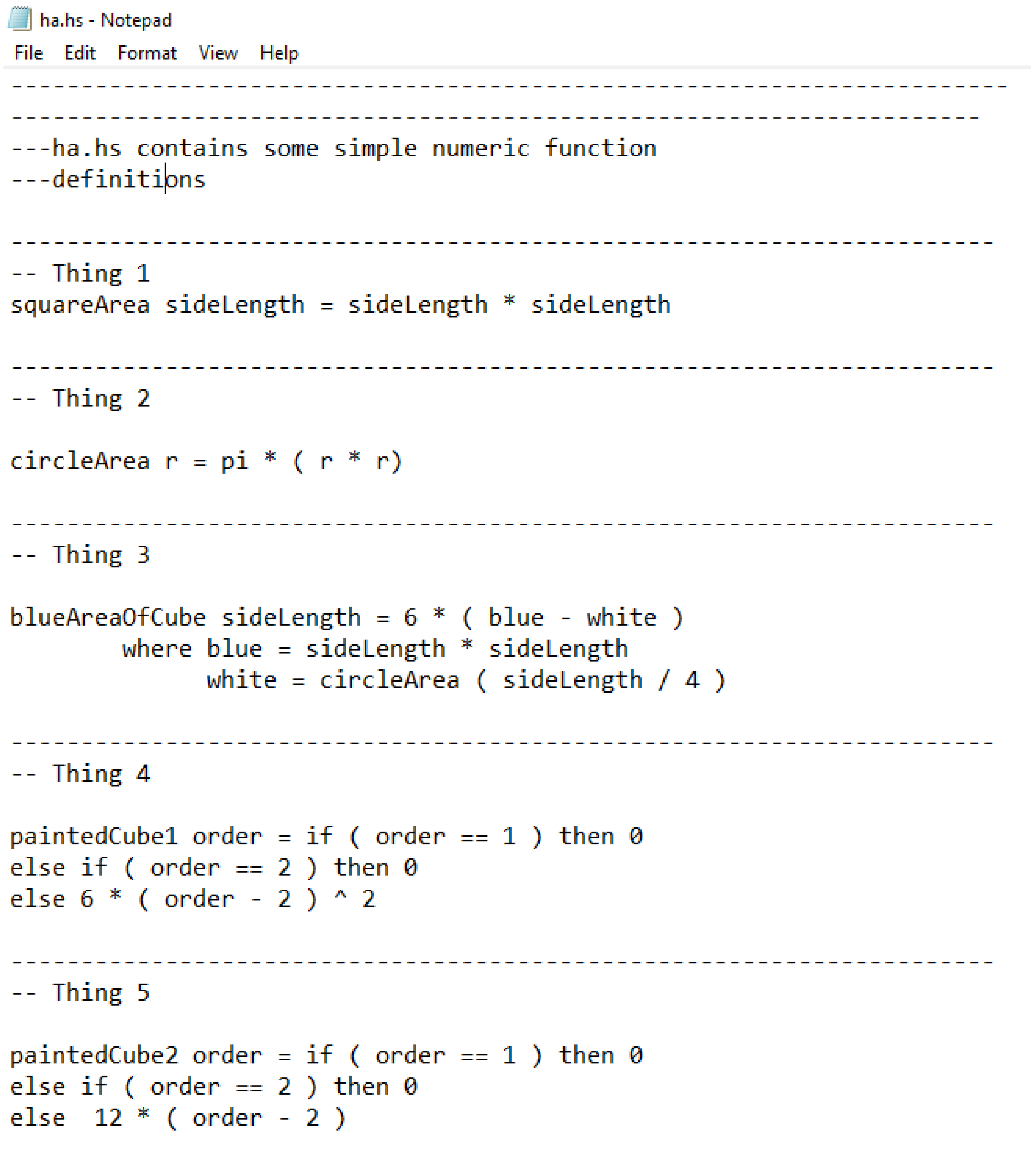
Abstract: In this assignment, I will learn basic Haskell by making some interesting programs in the language.

Task 1 Mindfully mimicking the demo:

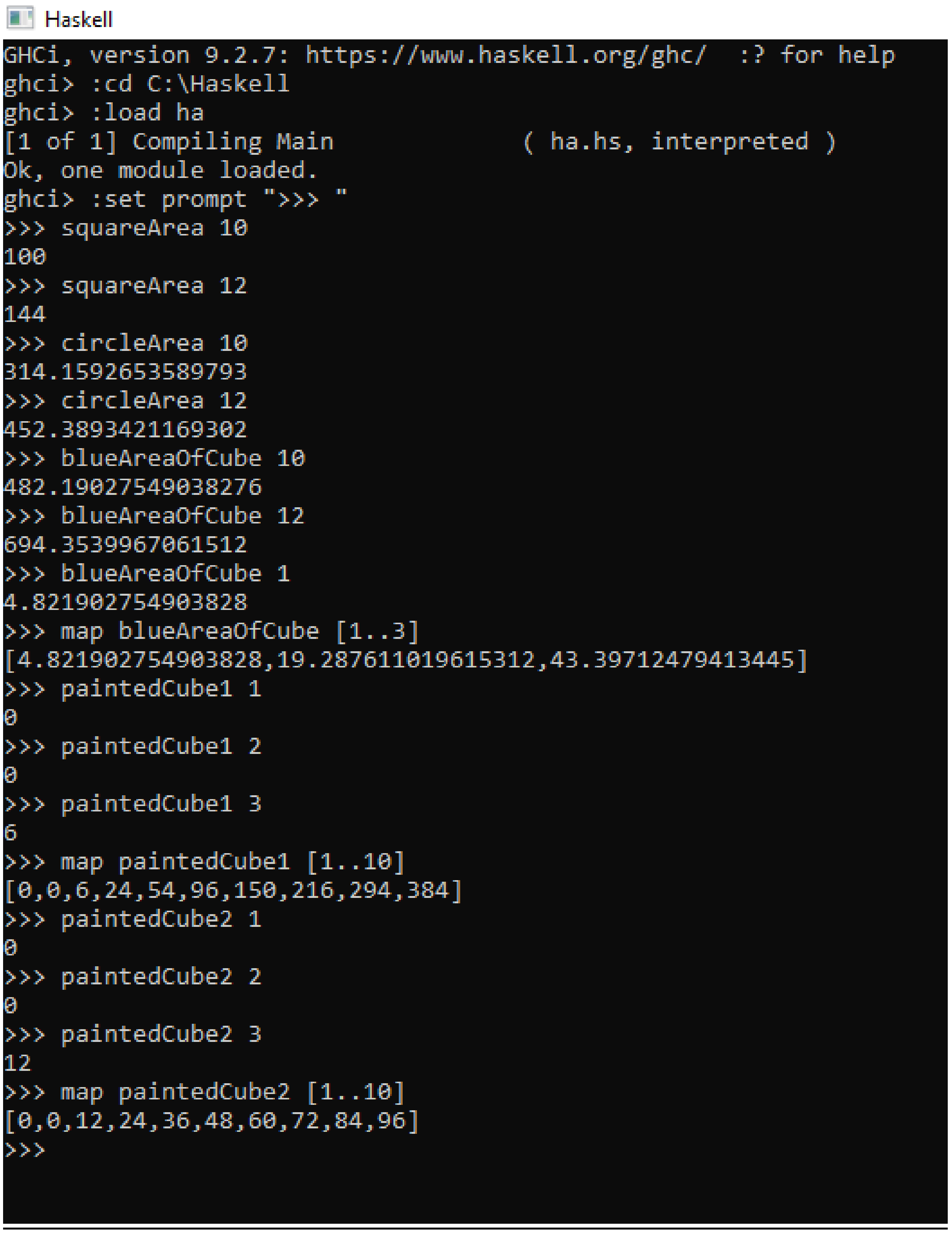


```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> length [2,3,5,7]
4
>>> words "need more coffee"
["need","more","coffee"]
>>> unwords ["need","more","coffee"]
"need more coffee"
>>> reverse "need more coffee"
"eeffoc erom deen"
>>> reverse ["need","more","coffee"]
["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>:15:13: error: lexical error at character '\8217'
>>> ( \x -> x /= ’’ ) ’Q’

<interactive>:16:14: error: lexical error at character '\8217'
>>> ( \x -> x /= ’’ ) ’Q’

<interactive>:17:14: error: lexical error at character '\8217'
>>> ( \x -> x /= '' ) 'Q'

<interactive>:18:14: error:
    Parser error on `''`
    Character literals may not be empty
>>> ( \x -> x /= ' ' ) 'Q'
True
>>> ( \x -> x /= ' ' ) ' '
False
>>> filter( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
>>>
```

Task 2 Numeric Function Definitions:



```
-----------------------------------------------------------------------
---------------------------------------------------------------------
---ha.hs contains some simple numeric function
---definitions

----------------------------------------------------------------------
-- Thing 1
squareArea sideLength = sideLength * sideLength

----------------------------------------------------------------------
-- Thing 2

circleArea r = pi * ( r * r)

----------------------------------------------------------------------
-- Thing 3

blueAreaOfCube sideLength = 6 * ( blue - white )
        where blue = sideLength * sideLength
              white = circleArea ( sideLength / 4 )

----------------------------------------------------------------------
-- Thing 4

paintedCube1 order = if ( order == 1 ) then 0
else if ( order == 2 ) then 0
else 6 * ( order - 2 ) ^ 2

----------------------------------------------------------------------
-- Thing 5

paintedCube2 order = if ( order == 1 ) then 0
else if ( order == 2 ) then 0
else  12 * ( order - 2 )
```

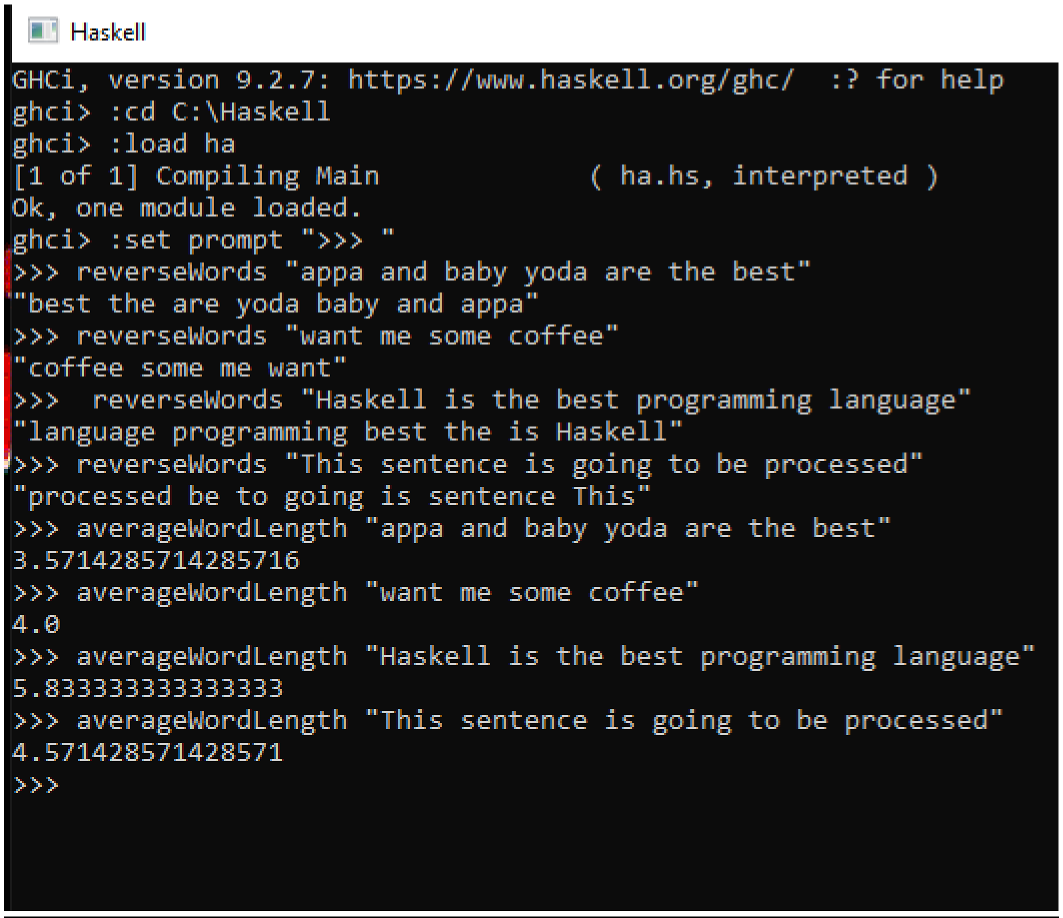


Haskell

```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :cd C:\Haskell
ghci> :load ha
[1 of 1] Compiling Main             ( ha.hs, interpreted )
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]
>>>
```

Task 3 Puzzlers:

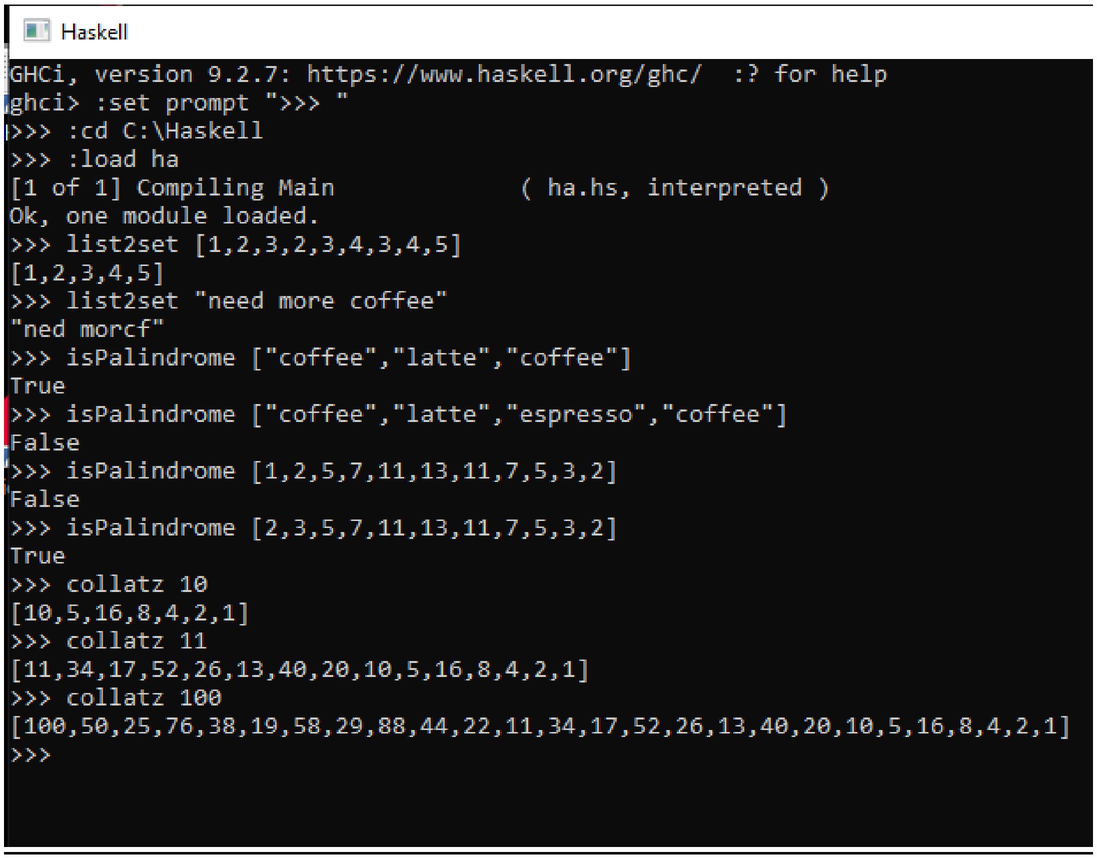
```
----------------------------------------------------------------------
-- Thing 6

reverseWords string = unwords reversedWordsArray
        where wordsArray = words string
              reversedWordsArray = reverse wordsArray


----------------------------------------------------------------------
-- Thing 7

averageWordLength string = ( stringLength / wordAmount )
        where wordsArray = words string
              wordAmount = fromIntegral ( length wordsArray )
              stringLength = fromIntegral ( length string  ) - ( wordAmount - 1 )
```



```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :cd C:\Haskell
ghci> :load ha
[1 of 1] Compiling Main             ( ha.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"
>>>  reverseWords "Haskell is the best programming language"
"language programming best the is Haskell"
>>> reverseWords "This sentence is going to be processed"
"processed be to going is sentence This"
>>> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
>>> averageWordLength "want me some coffee"
4.0
>>> averageWordLength "Haskell is the best programming language"
5.833333333333333
>>> averageWordLength "This sentence is going to be processed"
4.571428571428571
>>>
```

Task 4 Recursive List Processors:

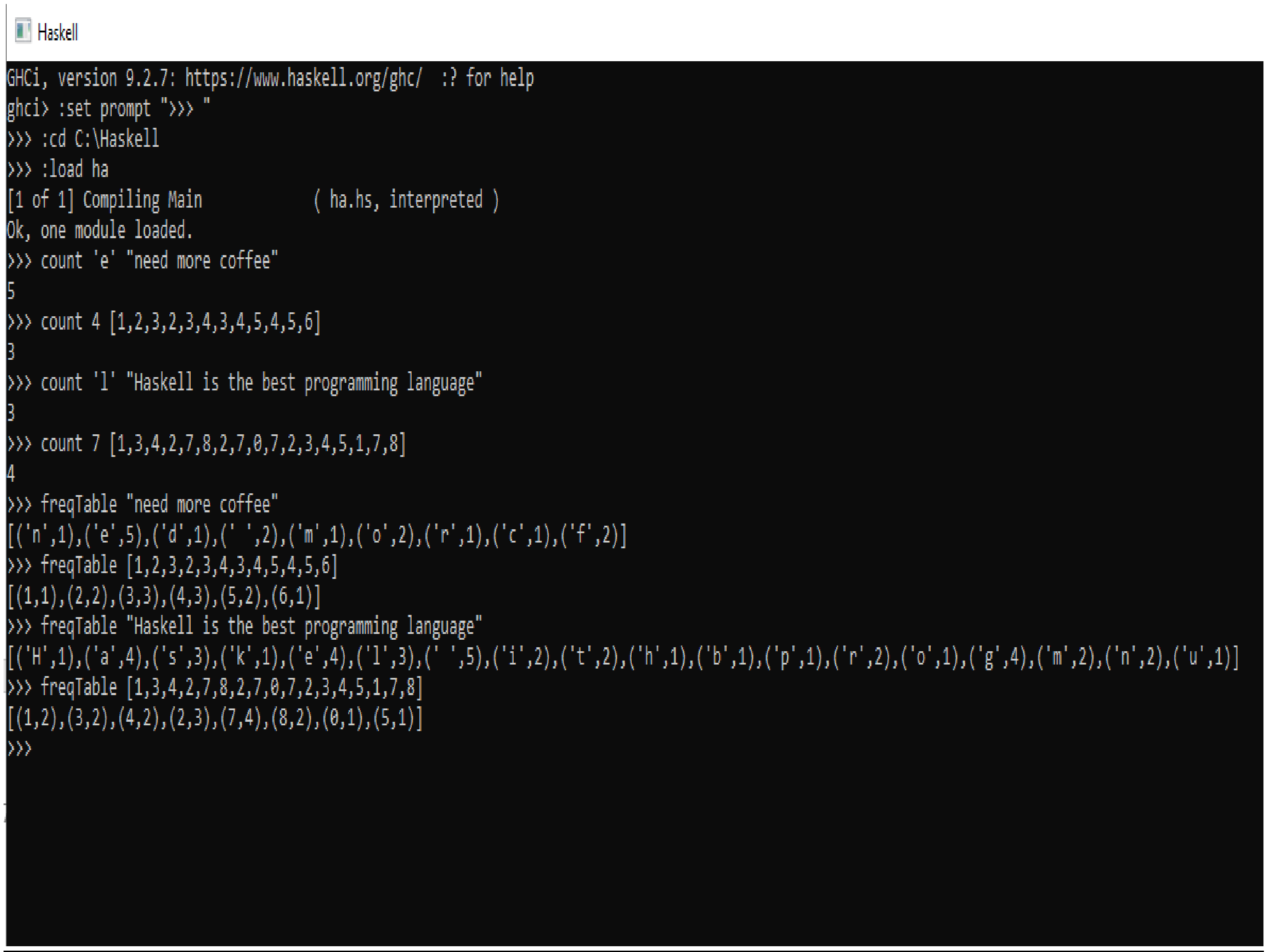
```
---------------------------------------------------------------------
-- Thing 8

list2set [] = []
list2set (x:xs) = x : list2set (filter ( /= x ) xs )

---------------------------------------------------------------------
-- Thing 9

isPalindrome [] = True
isPalindrome [_] = True
isPalindrome (x:xs)
   | x == last xs = isPalindrome (init xs)
   | otherwise    = False

---------------------------------------------------------------------
-- Thing 10

collatz 1 = [1]
collatz integer
   | even integer = integer : collatz ( integer `div` 2 )
   | otherwise = integer : collatz ( 3 * integer + 1 )
```


Haskell

```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :cd C:\Haskell
>>> :load ha
[1 of 1] Compiling Main             ( ha.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"
"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 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]
>>>
```

Task 5 List Comprehensions:

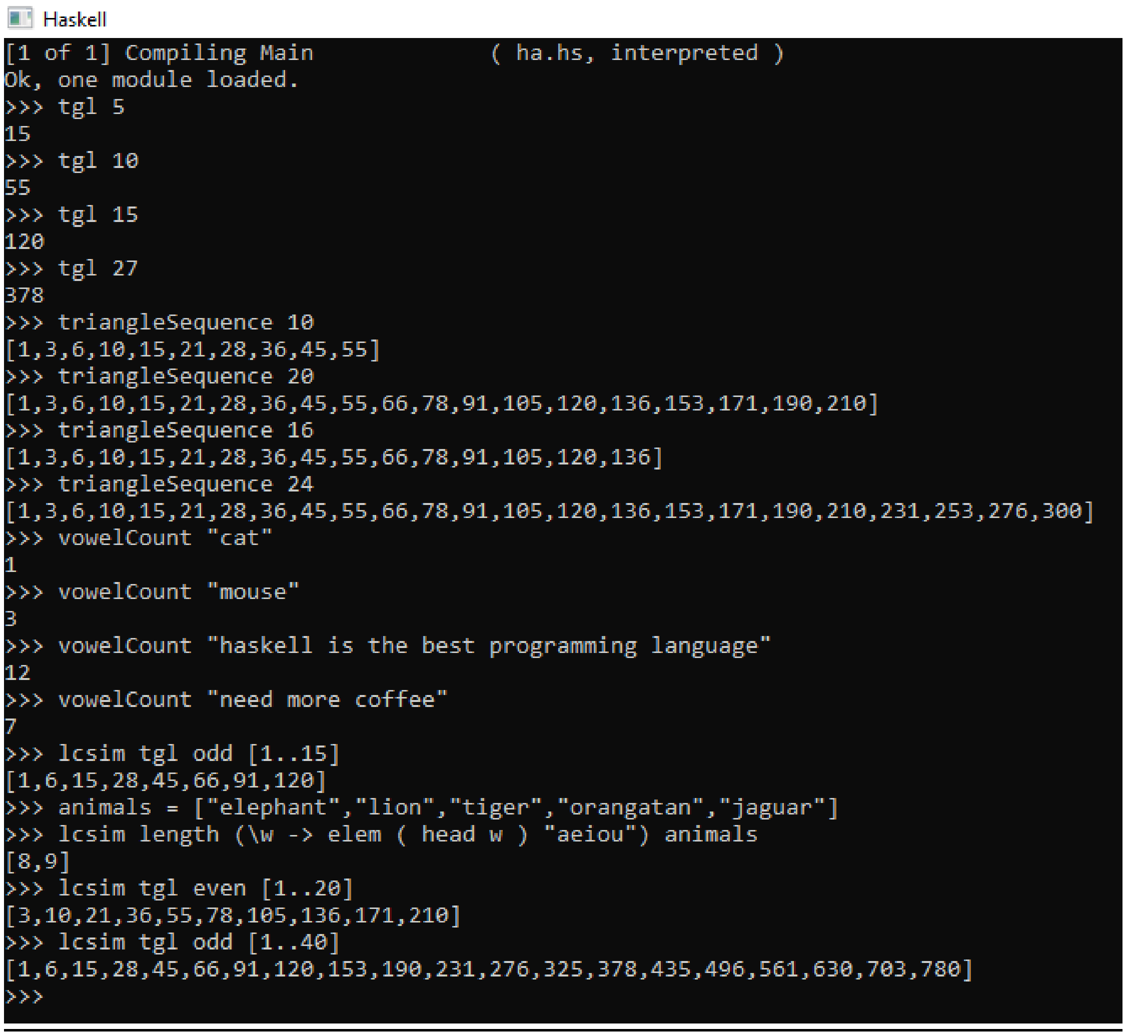
```
--------------------------------------------------------------------------
-- Thing 11

count object list = timesPresent
        where timesPresent = length [ x | x <- list, x == object ]

--------------------------------------------------------------------------
-- Thing 12

freqTable list = table
        where set = list2set list
              table = [ (x, count x list) | x <- set ]
```



```
Haskell
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :cd C:\Haskell
>>> :load ha
[1 of 1] Compiling Main             ( ha.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
>>> count 'l' "Haskell is the best programming language"
3
>>> count 7 [1,3,4,2,7,8,2,7,0,7,2,3,4,5,1,7,8]
4
>>> 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)]
>>> freqTable "Haskell is the best programming language"
[('H',1),('a',4),('s',3),('k',1),('e',4),('l',3),(' ',5),('i',2),('t',2),('h',1),('b',1),('p',1),('r',2),('o',1),('g',4),('m',2),('n',2),('u',1)]
>>> freqTable [1,3,4,2,7,8,2,7,0,7,2,3,4,5,1,7,8]
[(1,2),(3,2),(4,2),(2,3),(7,4),(8,2),(0,1),(5,1)]
>>>
```

Task 6 Higher Order Functions:

```
----------------------------------------------------------------------
-- Thing 13

tgl integer = foldl (+) 0 [1..integer]

----------------------------------------------------------------------
-- Thing 14

triangleSequence integer = map tgl [1..integer]

----------------------------------------------------------------------
-- Thing 15

vowelCount string = length ( filter (\x ->  x == 'a' || x == 'e' ||  x == 'i' ||  x == 'o' ||  x == 'u' ) string )

----------------------------------------------------------------------
-- Thing 16

lcsim func pred list =  map func ( filter ( \x -> pred x ) list )
```



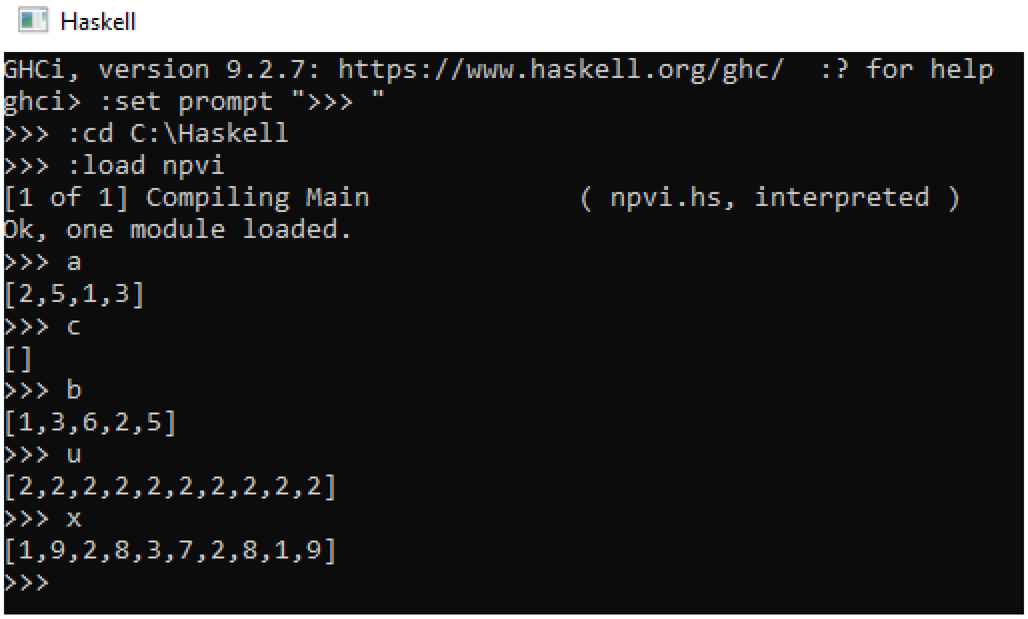
Haskell

```
[1 of 1] Compiling Main             ( ha.hs, interpreted )
Ok, one module loaded.
>>> tgl 5
15
>>> tgl 10
55
>>> tgl 15
120
>>> tgl 27
378
>>> 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 16
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136]
>>> triangleSequence 24
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> vowelCount "haskell is the best programming language"
12
>>> vowelCount "need more coffee"
7
>>> 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]
>>> lcsim tgl odd [1..40]
[1,6,15,28,45,66,91,120,153,190,231,276,325,378,435,496,561,630,703,780]
>>>
```

Task 7 An interesting statistic: nPVI:

Task 7a Test data:

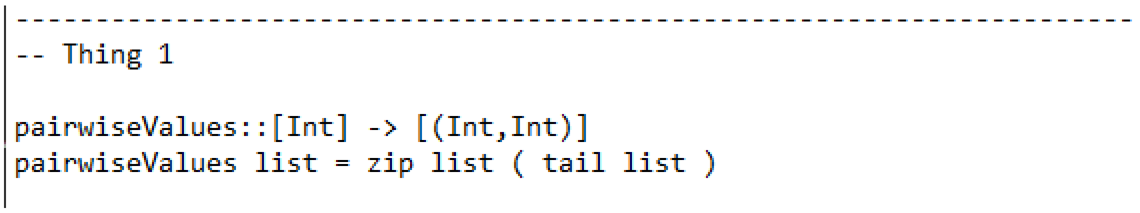
Note: The test data for c was not present in the specification document. Because of this, I made c just an empty array.



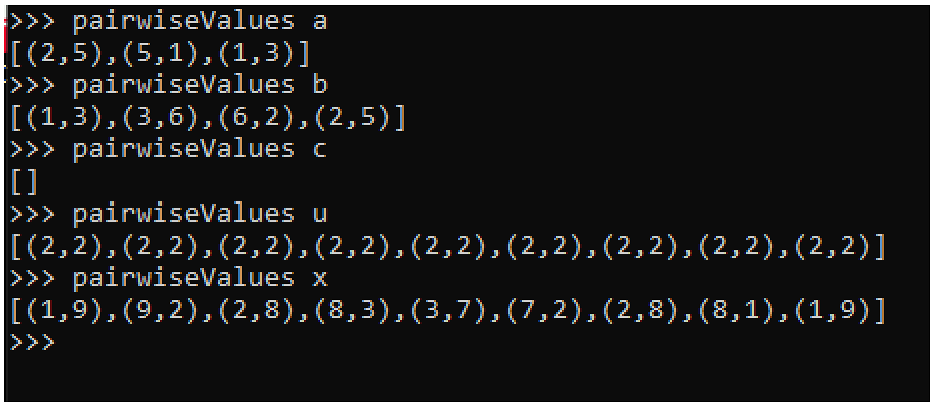
Haskell

```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :set prompt ">>> "
>>> :cd C:\Haskell
>>> :load npvi
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
>>> a
[2,5,1,3]
>>> c
[]
>>> b
[1,3,6,2,5]
>>> u
[2,2,2,2,2,2,2,2,2,2]
>>> x
[1,9,2,8,3,7,2,8,1,9]
>>>
```

Task 7b The pairwiseValues function:



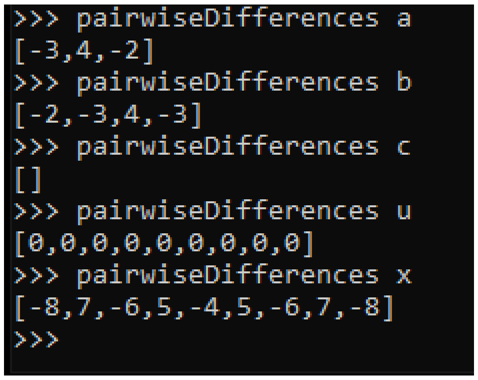
```
-----------------------------------------------------------------------
-- Thing 1

pairwiseValues::[Int] -> [(Int,Int)]
pairwiseValues list = zip list ( tail list )
```



```
>>> pairwiseValues a
[(2,5),(5,1),(1,3)]
>>> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
>>> pairwiseValues c
[]
>>> 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:

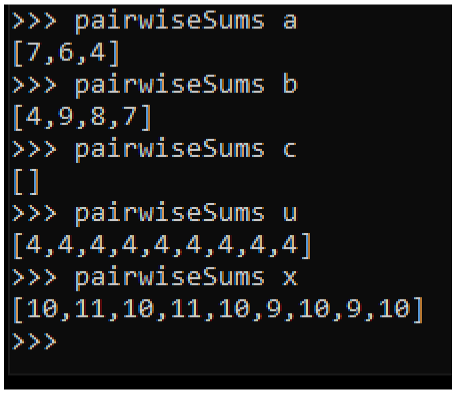
```
----------------------------------------------------------------------
-- Thing 2

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



```
>>> pairwiseDifferences a
[-3,4,-2]
>>> pairwiseDifferences b
[-2,-3,4,-3]
>>> pairwiseDifferences c
[]
>>> 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:

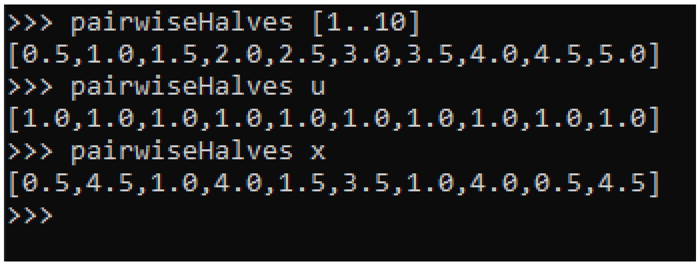
```
----------------------------------------------------------------------
-- Thing 3

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



```
>>> pairwiseSums a
[7,6,4]
>>> pairwiseSums b
[4,9,8,7]
>>> pairwiseSums c
[]
>>> pairwiseSums u
[4,4,4,4,4,4,4,4,4]
>>> pairwiseSums x
[10,11,10,11,10,9,10,9,10]
>>>
```

Task 7e The pairwiseHalves function:

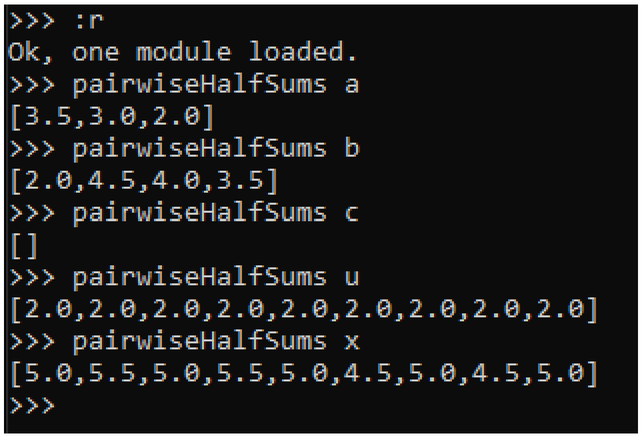
```
----------------------------------------------------------------------
-- Thing 4

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

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



```
>>> 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:

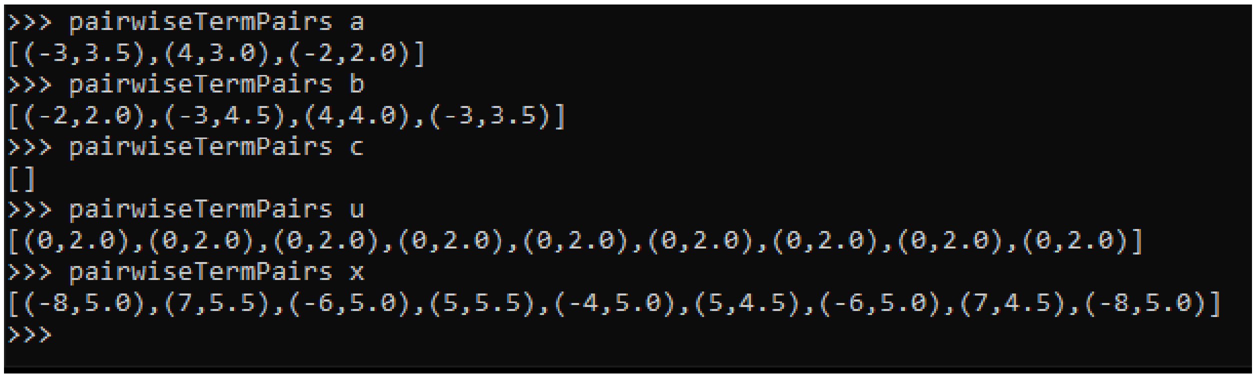
```
----------------------------------------------------------------------
-- Thing 5

pairwiseHalfSums::[Int] -> [Double]
pairwiseHalfSums list =   pairwiseHalves ( pairwiseSums list )
```



```
>>> :r
Ok, one module loaded.
>>> pairwiseHalfSums a
[3.5,3.0,2.0]
>>> pairwiseHalfSums b
[2.0,4.5,4.0,3.5]
>>> pairwiseHalfSums c
[]
>>> 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:

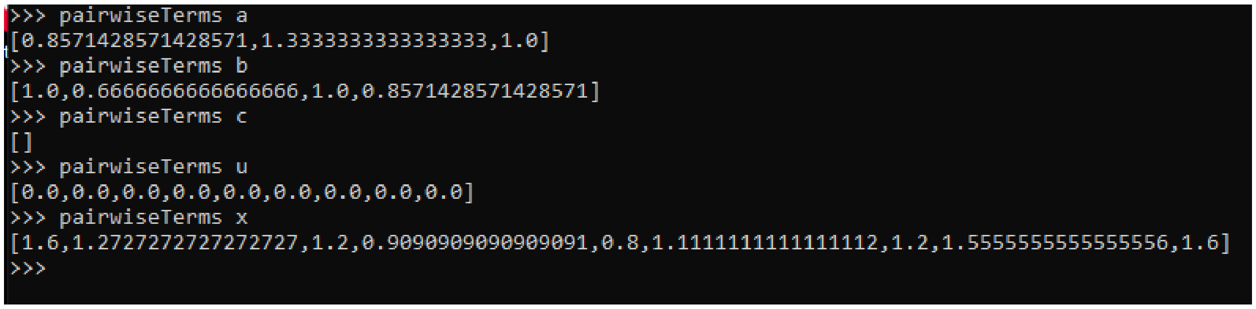
```
--------------------------------------------------------------------
-- Thing 6

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



```
>>> 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
[]
>>> 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)]
>>>
```

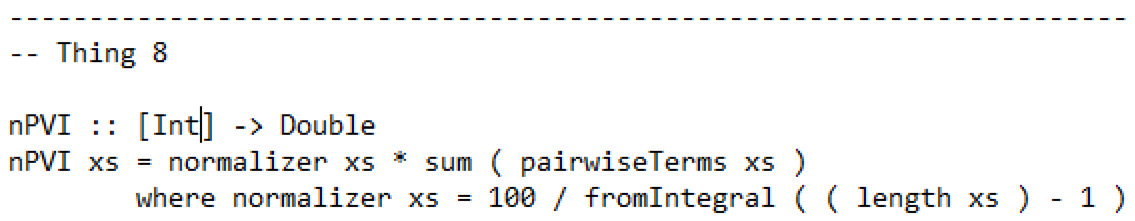
Task 7h The pairwiseTerms function:

```
--------------------------------------------------------------------
-- Thing 7

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

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

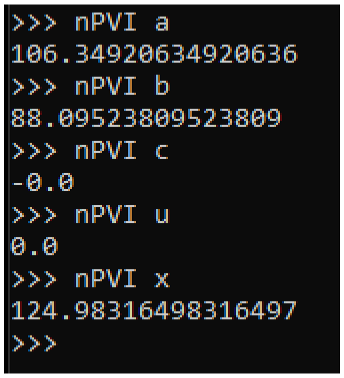


```
>>> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
>>> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
>>> pairwiseTerms c
[]
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.5555555555555556,1.6]
>>>
```

Task 7i The nPVI function:



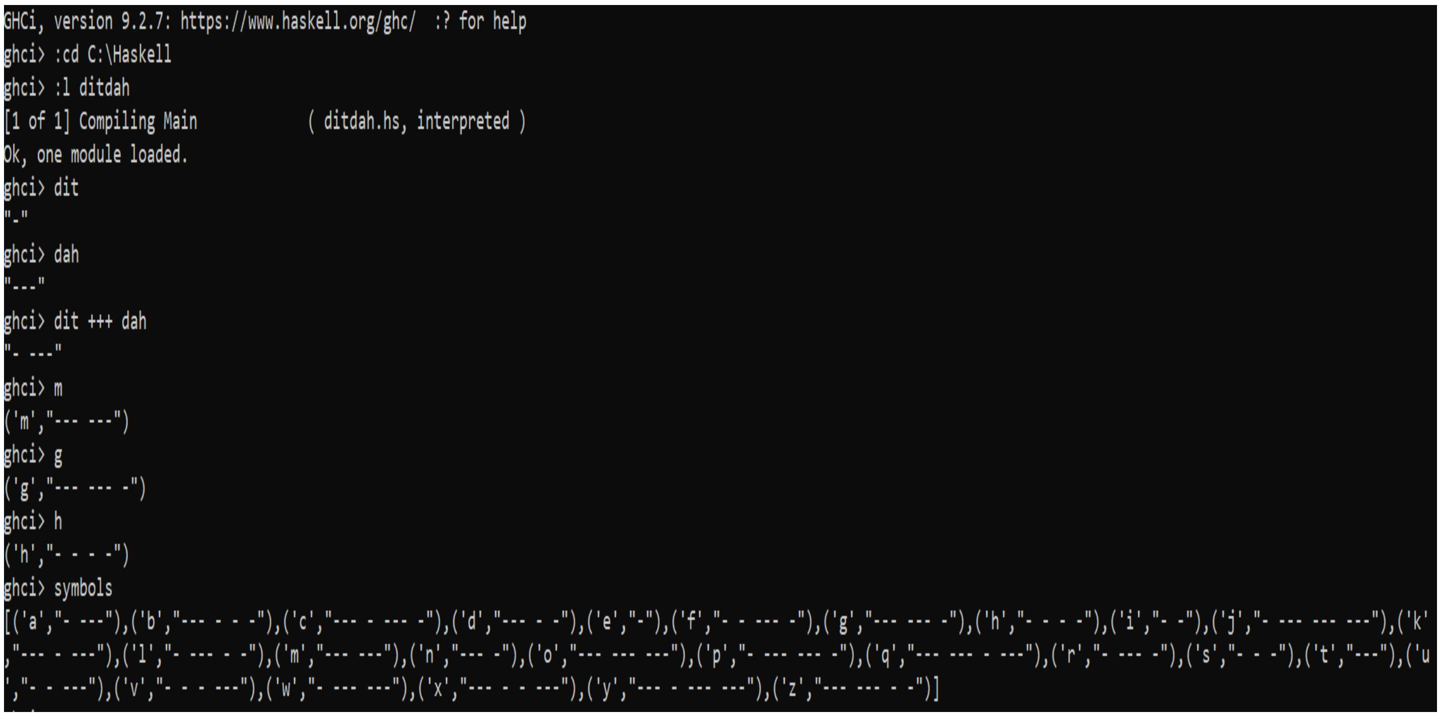
```
--------------------------------------------------------------------
-- Thing 8

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



```
>>> nPVI a
106.34920634920636
>>> nPVI b
88.09523809523809
>>> nPVI c
-0.0
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
>>>
```

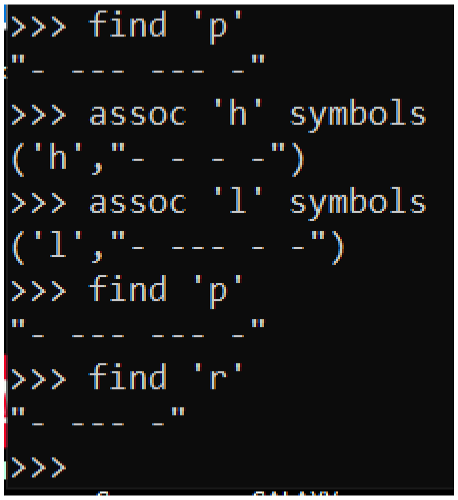
Task 8 Historic code: The dit dah code:

Subtask 8a:



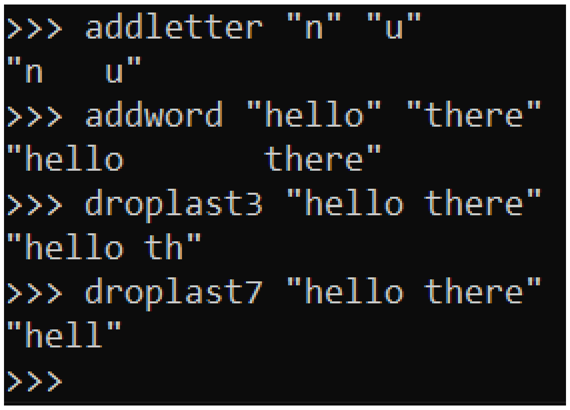
```
GHCi, version 9.2.7: https://www.haskell.org/ghc/  :? for help
ghci> :cd C:\Haskell
ghci> :l ditdah
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
ghci> dit
"-"
ghci> dah
"---"
ghci> dit +++ dah
"- ---"
ghci> m
('m',"--- ---")
ghci> g
('g',"--- --- -")
ghci> h
('h',"- - - -")
ghci> symbols
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -"),('d',"--- - -"),('e',"-"),('f',"- - --- -"),('g',"--- --- -"),('h',"- - - -"),('i',"- -"),('j',"- --- --- ---"),('k'
,"--- - ---"),('l',"- --- - -"),('m',"--- ---"),('n',"--- -"),('o',"--- --- ---"),('p',"- --- --- -"),('q',"--- --- - ---"),('r',"- --- -"),('s',"- - -"),('t',"---"),('u
',"- - ---"),('v',"- - - ---"),('w',"- --- ---"),('x',"--- - - ---"),('y',"--- - --- ---"),('z',"--- --- - -")]
```

Subtask 8b:



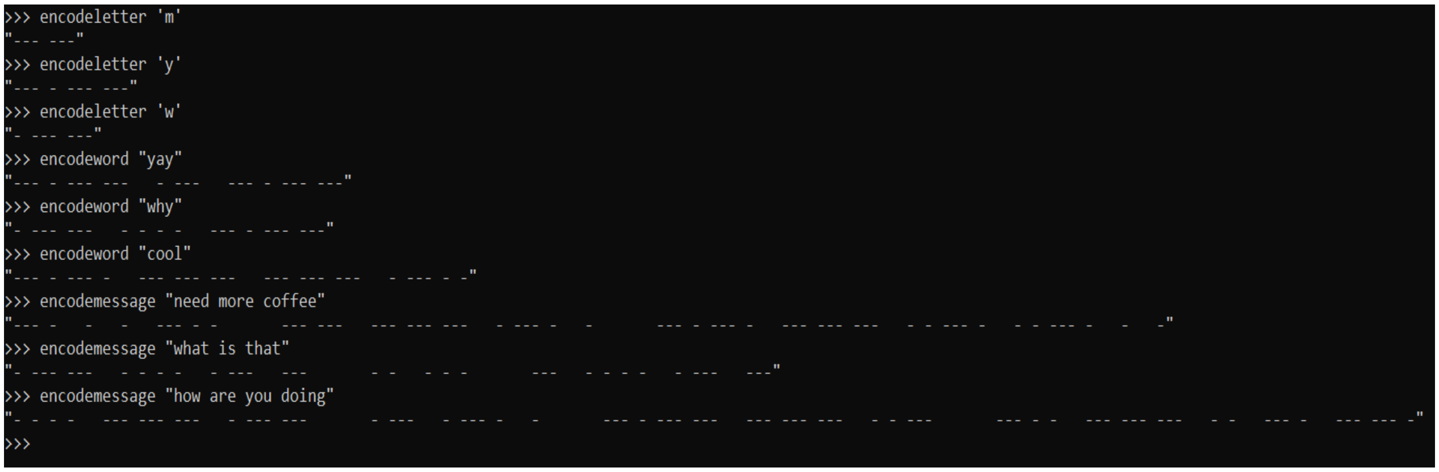
```
>>> find 'p'
"- --- --- -"
>>> assoc 'h' symbols
('h',"- - - -")
>>> assoc 'l' symbols
('l',"- --- - -")
>>> find 'p'
"- --- --- -"
>>> find 'r'
"- --- -"
>>>
```

Subtask 8c:



```
>>> addletter "n" "u"
"n   u"
>>> addword "hello" "there"
"hello       there"
>>> droplast3 "hello there"
"hello th"
>>> droplast7 "hello there"
"hell"
>>>
```

Subtask 8d:



```
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'y'
"--- - --- ---"
>>> encodeletter 'w'
"- --- ---"
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "why"
"- --- ---   - - - -   --- - --- ---"
>>> encodeword "cool"
"--- - --- -   --- --- ---   --- --- ---   - --- - -"
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - --- -   -       --- - --- -   --- --- ---   - - --- -   - - --- -   -   -"
>>> encodemessage "what is that"
"- --- ---   - - - -   - ---   ---       - -   - - -       ---   - - - -   - ---   ---"
>>> encodemessage "how are you doing"
"- - - -   --- --- ---   - --- ---       - ---   - --- -   -       --- - --- ---   --- --- ---   - - ---       --- - -   --- --- ---   - -   --- -   --- --- -"
>>>
```