Haskell Programming Assignment: Various Computations

◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇

Learning Abstract: For my first Haskell Programming Assignment, there are eight tasks in total which are based on basic functions, recursive list processing, higher order functions, and list comprehensions in Haskell.

— —

Task 1 - Mindfully Mimicking the Demo

```
Last login: Sat Nov 26 11:07:29 on ttys000
[jdvargas@JDs-MacBook-Air ~ % ghci
GHCi, version 9.2.5: https://www.haskell.org/ghc/  :? fo
[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'
True
[>>> ( \x -> x /= ' ') ' '
False
[>>> filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
[>>> :quit
Leaving GHCi.
```

Task 2 - Numeric Function Definitions

Code:

```
squareArea number = number * number
circleArea radius = pi * radius^2
blueAreaOfCube edge = 6 * ((squareArea edge) - (circleArea (edge / 4)))
paintedCube1 order = if (order > 2) then (6 * ((order - 2)^2)) else 0
paintedCube2 order = if (order > 2) then (6 * (2 * (order - 2))) else 0
```

Demo:

```
ghci> :load whatever.hs
[1 of 1] Compiling Main             ( whatever.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
```

```
[>>> paintedCube 3

 <interactive>:14:1: error:
     • Variable not in scope: paintedCube :: t0 -> t
     • Perhaps you meant one of these:
         'paintedCube1' (line 4), 'paintedCube2' (line 5)
[>>> 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
```

Task 3 - Puzzlers

Code:

```
reverseWords string = unwords (reverse (words string ))
averageWordLength string = fromIntegral ( wordLength - numOfSpaces) / fromIntegral numOfWords
 where
  wordLength = length string
  numOfWords = length (words string)
  numOfSpaces = numOfWords -1
```

Demo:

```
ghci> :load whatever.hs
[1 of 1] Compiling Main             ( whatever.hs, interpreted )
Ok, one module loaded.
ghci> reverseWords "appa and baby yoda are the best"
"best the are yoda baby and appa"
ghci> reverseWords "want me some coffee"
"coffee some me want"
ghci> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
ghci> averageWordLength "want me some coffee"
4.0
ghci> quit
```

Task 4 - Recursive List Processors

Code:

```
listingsets[] = []
listingsets ( x : xy ) = if ( elem x xy ) then ( somethingElse ) else ( x : somethingElse )
 where
  somethingElse = listingsets xy

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

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

Demo:

```
[ghci> :load whatever.hs
 [1 of 1] Compiling Main             ( whatever.hs, interpreted )
 Ok, one module loaded.
[ghci> listingsets [1,2,3,2,3,4,3,4,5]
 [1,2,3,4,5]
[ghci> listingsets "need more coffee"
 "ndmr cofe"
[ghci> isPalindrome ["coffee","latte","coffee"]
 True
[ghci> isPalindrome[1,2,5,7,11,13,11,7,5,3,2]
 False
[ghci> isPalindrome[2,3,5,7,11,13,11,7,5,3,2]
 True
[ghci> collatz 10
 [10,5,16,8,4,2,1]
[ghci> collatz 11
 [11,34,17,52,26,13,40,20,10,5,16,8,4,2,1]
[ghci> 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]
[ghci> :quit
 Leaving GHCi.
```

Task 5 - List Comprehensions

Code:

```
listingsets[] = []
listingsets ( x : xy ) = if ( elem x xy ) then ( somethingElse ) else ( x : somethingElse )
 where
  somethingElse = listingsets xy

count element listOfElements = length [x | x<- listOfElements, x == element]

freqTable listOfElements = zip firstList secondList
 where
  firstList = [x | x<- (listingsets listOfElements)]
  secondList = [count y listOfElements | y <- (listingsets listOfElements)]
```

Demo:

```
[>>> 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

Code:

```
tgl number = foldl (+) 0 [1..number]

triangleSequence number = map (tgl) [1..number]

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

lcsim mapFunction predFilter listOfElements = map (mapFunction) (filter (predFilter) listOfElements)
```

Demo:

```
ghci> :load whatever.hs
[1 of 1] Compiling Main             ( whatever.hs, interpreted )
Ok, one module loaded.
ghci> tgl 5
15
ghci> tgl 10
55
ghci> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
ghci> triangleSequence 20
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210]
ghci> vowelCount "cat"
1
ghci> vowelCount "mouse"
3
ghci> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
ghci> animals = ["elephant","lion","tiger","orangatan","jaguar"]
ghci> lcsim length (\w -> elem (head w ) "aeiou") animals
[8,9]
ghci> :quit
Leaving GHCi.
```

Task 7 - An Interesting Statistic: nPVI

Task 7a-Test Data
Code And Demo:

```
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]
```

```
ghci> :load whatever.hs
[1 of 1] Compiling Main
Ok, one module loaded.
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> :quit
Leaving GHCi.
```

Task 7b - The pairwiseValues function

Code and Demo:

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

```
ghci> :load whatever.hs
[1 of 1] Compiling Main             ( whatever.hs, inter
Ok, one module loaded.
ghci> pairwiseValues a
[(2,5),(5,1),(1,3)]
ghci> pairwiseValues b
[(1,3),(3,6),(6,2),(2,5)]
ghci> pairwiseValues c
[(4,4),(4,2),(2,1),(1,1),(1,2),(2,2),(2,4),(4,4),(4,8)]
ghci> pairwiseValues u
[(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2),(2,2)]
ghci> pairwiseValues x
[(1,9),(9,2),(2,8),(8,3),(3,7),(7,2),(2,8),(8,1),(1,9)]
ghci>
```

Task 7c - The pairwiseDifferences function

Code:

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

pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences listOfNumbers = map (\ (a,b) -> a - b) (pairwiseValues listOfNumbers)
```

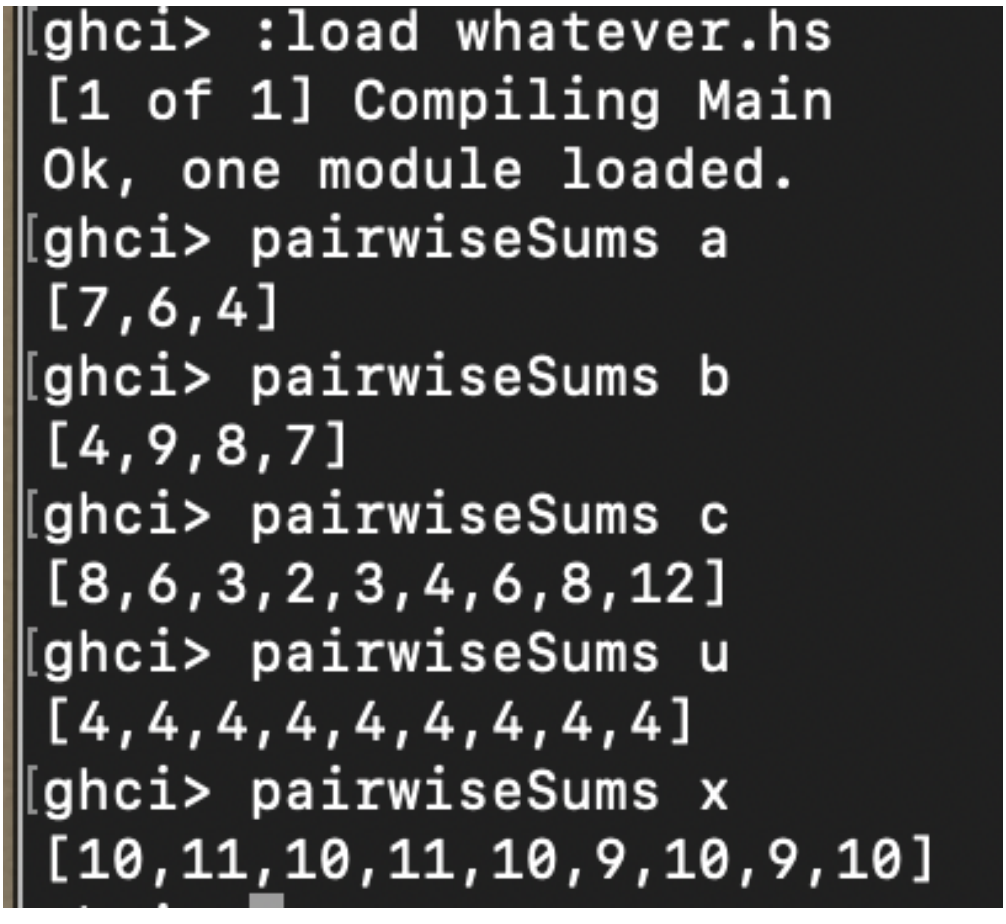
```
ghci> :load whatever.hs
[1 of 1] Compiling Main
Ok, one module loaded.
ghci> pairwiseDifferences a
[-3,4,-2]
ghci> pairwiseDifferences b
[-2,-3,4,-3]
ghci> pairwiseDifferences c
[0,2,1,0,-1,0,-2,0,-4]
ghci> pairwiseDifferences u
[0,0,0,0,0,0,0,0,0]
ghci> pairwiseDifferences x
[-8,7,-6,5,-4,5,-6,7,-8]
```

Task 7d - The pairwiseSums function

Code:

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

pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences listOfNumbers = map (\ (a,b) -> a - b) (pairwiseValues listOfNumbers)

pairwiseSums :: [Int] -> [Int]
pairwiseSums listOfNumbers = map (\ (a,b) -> a + b) (pairwiseValues listOfNumbers)
```

Demo:



Task 7e - The pairwiseHalves function

Code:

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

pairwiseDifferences :: [Int] -> [Int]
pairwiseDifferences listOfNumbers = map (\ (a,b) -> a - b) (pairwiseValues listOfNumbers)

pairwiseSums :: [Int] -> [Int]
pairwiseSums listOfNumbers = map (\ (a,b) -> a + b) (pairwiseValues listOfNumbers)

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

pairwiseHalves :: [Int] -> [Double]
pairwiseHalves listOfNumbers = map half listOfNumbers
```

DEMO:

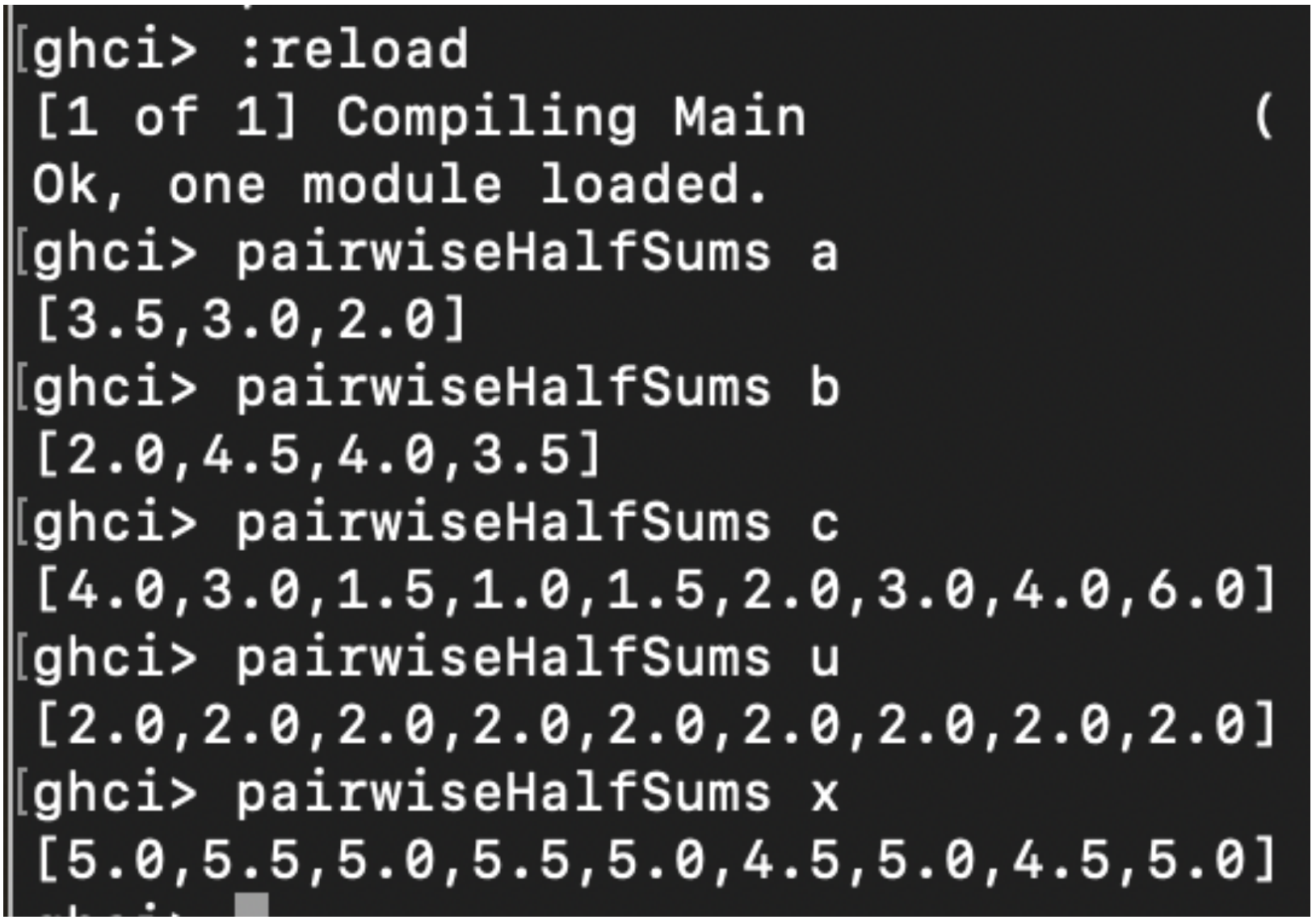
```
ghci> :load whatever.hs
 [1 of 1] Compiling Main             ( whatev
 Ok, one module loaded.
ghci> pairwiseHalves [1..10]
 [0.5,1.0,1.5,2.0,2.5,3.0,3.5,4.0,4.5,5.0]
ghci> pairwiseHalves u
 [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
ghci> pairwiseHalves x
 [0.5,4.5,1.0,4.0,1.5,3.5,1.0,4.0,0.5,4.5]
```

Task 7f - The pairwiseHalfSums function

Code:

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

Demo:



Task 7g - The pairwiseTermPairs function

Code:

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

Demo:

```
ghci> :reload
[1 of 1] Compiling Main             ( whatever.hs, interpreted )
Ok, one module loaded.
ghci> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
ghci> pairwiseTermPairs b
[(-2,2.0),(-3,4.5),(4,4.0),(-3,3.5)]
ghci> 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)]
ghci> 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)]
ghci> 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

Code:

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

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

Demo:

```
ghci> :reload
[1 of 1] Compiling Main             ( whatever.hs, interpreted )
Ok, one module loaded.
ghci> pairwiseTerms a
[0.8571428571428571,1.3333333333333333,1.0]
ghci> pairwiseTerms b
[1.0,0.6666666666666666,1.0,0.8571428571428571]
ghci> pairwiseTerms c
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.66666666
666
ghc
[0.
ghc
[1.
555
555
```

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

Task 7i - The nPVI function

Code:

Demo:

```
ghci> :reload
[1 of 1] Compiling Main
Ok, one module loaded.
ghci> nPVI a
106.34920634920636
ghci> nPVI b
88.09523809523809
ghci> nPVI c
37.03703703703703
ghci> nPVI u
0.0
ghci> nPVI x
124.98316498316497
```

Task 8 - Historic Code: The Dit Dah Code

Subtask 8a

```
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

```
[1 of 1] Compiling Main
Ok, one module loaded.
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"
"___ _ ___ ___   _ ___   ___ _ ___ ___"
ghci> encodeword "twelve"
"___   _ ___ ___   _   _ ___ _ _   _ _ _ ___   _"
ghci> encodeword "you"
"___ _ ___ ___   ___ ___ ___   _ _ ___"
ghci> encodemessage "need more coffee"
"___ _   _   _   ___ _ _       ___ ___   ___ ___ ___   _ ___ _   _      ___ _ ___ _   ___ ___ ___
   _ _ ___ _   _ _ ___ _   _   _"
ghci> encodemessage "hi my name is jd"
"_ _ _ _   _ _       ___ ___   ___ _ ___ ___      ___ _   _ ___   ___ ___   _      _ _   _ _ _
     _ ___ ___ ___   ___ _ _"
ghci> encodemessage "i like pizza"
"_ _       _ ___ _ _   _ _   ___ _ ___   _      _ ___ ___ _   _ _   ___ ___ _ _   ___ ___ _ _   _
 ___"
```