
Haskell Programming Assignment: Various Computations

Abstract: This is my last assignment for the semester aside from the essay on Rust. This is the only assignment which I do in Haskell. This assignment focuses on getting acquitted with Haskell. I spend most of this assignment writing functions which solve certain problems. For example in task two there is the painted square problem and I am trying to find out how many sides of a square cut up into x amount of slices will have a single blue face. Then in Task three I am messing with Strings and Char arrays to reverse sentences. Ultimately the tasks in this assignment are based on many of the different tasks I've done earlier this semester in Lisp and Prolog, therefore there is a lot of variation. I enjoyed the last task the most, because I got to mess with morse code. Morse code is very well known but if you don't know what morse code is, it's a very famous code that was originally used to send encoded messages over telegraph.

Task 1 - Mindfully Mimicking the Demo

```
C:\Users\roses>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"]
>>> 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 x = x * x
circleArea r = pi * (r * r)
blueAreaOfCube side = (6 * squareArea side ) - (circleArea (side *
1/4)* 6)
paintedCube1 count = if (count == 1 ) then 0
     else 6 * (count - 2) * (count - 2)
paintedCube2 count =if (count == 1 ) then 0
     else 12 * (count - 2)
```

Task 2 - Numeric Function Definitions [**DEMO**]

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
>>> :cd Desktop
>>> :cd luc
>>> :load numericfun.hs
[1 of 1] Compiling Main             ( numericfun.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
>>> blueAreaOfCube [1..3]
```

```
>>> 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 [**CODE**]

```
reverseWords :: String -> String
reverseWords x = (unwords (reverse  (words x)))

averageWordLength lst = fromIntegral (sum (map length (words
lst)))/amount
   where amount = fromIntegral (length (words lst))
```

Task 3 - Puzzlers [**DEMO**]

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :set prompt ">>> "
>>> :cd Desktop
>>> :cd luc
>>> :load puzzle.hs
[1 of 1] Compiling Main             ( puzzle.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"
```

```
>>> reverseWords "give me sushi and soup"
"soup and sushi me give"
>>> reverseWords "this took forever to write"
"write to forever took this"
>>> averageWordLength "appa and baby yoda are the best"
3.5714285714285716
>>> averageWordLength "want me some coffee"
4.0
>>> averageWordLength "no one said to use the same sentence"
3.625
>>> averageWordLength "long average give to meeeeeeeeeeeeee"
6.6
```

Task 4 - Recursive List Processors [CODE]

```
list2set [] = []
list2set (x:xs) = if (elem x xs) then list2set xs
   else x : list2set xs

isPalindrome :: (Eq a) => [a] -> Bool
isPalindrome [] = True
isPalindrome [x] = True
isPalindrome lst = if ((head lst) == (last lst)) then isPalindrome
(tail(init lst))
   else False

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

Task 4 - Recursive List Processors [DEMO]

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :cd Desktop
Prelude> :cd luc
Prelude> :set prompt ">>> "
>>> :load listprocess.hs
[1 of 1] Compiling Main             ( listprocess.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]
>>>
```

Task 5 - List Comprehensions **[CODE]**

```
list2set [] = []
list2set (x:xs) = if (elem x xs) then list2set xs
   else x : list2set xs

count obj lst = length [ x | x <- lst, obj == x ]

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

Task 5 - List Comprehensions **[DEMO]**

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :cd Desktop
Prelude> :cd luc
Prelude> :load listcomp.hs
[1 of 1] Compiling Main             ( listcomp.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> count 'e' "need more coffee"
5
>>> count 4 [1,2,3,2,3,4,3,4,5,4,5,6]
3
>>> count 'i' "hi you are tired"
2
>>> count 8 [8,8,8,7,7,8,8,9,8,0]
6
```

```
>>> 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)]
>>> freqTable "hi you are tired"
[('h',1),('y',1),('o',1),('u',1),('a',1),(' ',3),('t',1),('i',2),('r',2),('e',2),('d',1)]
>>> freqTable [8,8,8,7,7,8,8,9,8,0]
[(7,2),(9,1),(8,6),(0,1)]
>>>
```

Task 6 - Higher Order Functions **[CODE]**

```
tgl :: Int -> Int
tgl n = if (n == 1) then n + 0
   else foldl (+) n [tgl (n - 1)]

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

vowelCount :: [Char] -> Int
vowelCount word = length (filter (\x ->  x == 'a' || x == 'i' || x ==
'o' || x == 'u' || x == 'e')  word)

lcsim fun1 fun2 lst = (map (fun1) (filter (fun2) lst))
```

Task 6 - Higher Order Functions **[DEMO]**

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :cd Desktop
Prelude> :cd luc
Prelude> :load higher.hs
[1 of 1] Compiling Main             ( higher.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> tgl 5
15
>>> tgl 10
55
>>> tgl 7
28
>>> tgl 8
36
>>>
```

```
>>> 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 25
[1,3,6,10,15,21,28,36,45,55,66,78,91,105,120,136,153,171,190,210,231,253,276,300,325]
>>> triangleSequence 5
[1,3,6,10,15]
>>>
```

```
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> vowelCount "Matilda"
3
>>> vowelCount "hangry"
1
```

```
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> lcsim tgl even [2..20]
[3,10,21,36,55,78,105,136,171,210]
>>>  animals = ["elephant","lion","tiger","orangatan","jaguar"]
>>> lcsim length (\w -> elem ( head w ) "aeiou") animals
[8,9]
```

```
>>> names = ["Tony","Jack","Timmy","Red","Luci","Toby"]
>>> lcsim length (\w -> elem ( head w ) "T") names
[4,5,4]
```

Task 7 - An Interesting Statistic: nPVI

Task 7 **[ORIGINAL CODE]**

```
-- npvi.hs code. This is for the nPVI statistic function.
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 7a - Test data [**Demo**]

```
Prelude> :load npvi.hs
[1 of 1] Compiling Main             ( npvi.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]
>>>
```

Task 7b - The pairwiseValues [**Demo**]

```
C:\Users\roses>ghci
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :cd Desktop
Prelude> :cd luc
Prelude> :load npvi.hs
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> 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 - The pairwiseDifferences [**DEMO**]

```
>>> :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 - The pairwiseSums function [**DEMO**]

```
>>> :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 - The pairwiseHalves function [**DEMO**]

```
[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]
>>> half 4
2.0
>>>
```

Task 7f - The pairwiseHalfSums function [**DEMO**]

```
Prelude> :load npvi.hs
[1 of 1] Compiling Main             ( npvi.hs, interpreted )
Ok, one module loaded.
*Main> :set prompt ">>> "
>>> 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 – The pairwiseTermPairs function [**DEMO**]

```
>>> :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 – The pairwiseTerms function [**DEMO**]

```
>>> :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
[0.0,0.6666666666666666,0.6666666666666666,0.0,0.6666666666666666,0.0,0.6666666
666666666,0.0,0.6666666666666666]
```

```
>>> pairwiseTerms u
[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
>>> pairwiseTerms x
[1.6,1.2727272727272727,1.2,0.9090909090909091,0.8,1.1111111111111112,1.2,1.555
5555555555556,1.6]
```

```
>>> term (1,1.3)
0.7692307692307692
```

Task 7i - The nPVI function **[DEMO]**

```
>>> :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
>>>
```

IMPORTANT NOTE: Me testing the term and half functions are mixed in with the demos in which the function is used.

Task 7 Finished **[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 (x:xs) = zip (x:xs) xs

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

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

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

pairwiseHalves :: [Int] -> [Double]
pairwiseHalves lst = map half lst

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

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

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

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

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

Task 8 - Historic Code: The Dit Dah Code

Subtask 8a **[DEMO]**

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

Subtask 8b [**DEMO**]

```
GHCi, version 8.10.7: https://www.haskell.org/ghc/  :? for help
Prelude> :cd Desktop
Prelude> :cd luc
Prelude> :set prompt ">>> "
>>> :load ditdah.hs
[1 of 1] Compiling Main             ( ditdah.hs, interpreted )
Ok, one module loaded.
>>> assoc 'w' symbols
('w',"- --- ---")
>>> assoc 's' symbols
('s',"- - -")
>>> find 'e'
"-"
>>> find 'i'
"- -"
>>>
```

Subtask 8c [**DEMO**]

```
>>> addletter (encodeletter 'x') (encodeletter 'z')
"--- - - ---   --- --- - -"
```

```
>>> addword (encodeword "wombo") (encodeword "word")
"- --- ---   --- --- ---   --- ---   --- - - -   --- -
-- ---       - --- ---   --- --- ---   - --- -   --- -
 -"
```

```
>>> droplast3 [a,b,c,d,e,f]
[('a',"- ---"),('b',"--- - - -"),('c',"--- - --- -")]
```

```
>>> droplast7 [h,u,m,a,n,i,t,y]
[('h',"- - - -")]
```

Subtask 8d **[DEMO]**

```
>>> encodeletter 'm'
"--- ---"
>>> encodeletter 'v'
"- - - ---"
>>> encodeletter 'o'
"--- --- ---"
>>> encodeword "yay"
"--- - --- ---   - ---   --- - --- ---"
>>> encodeword "puppies"
"- --- --- -   - - ---   - --- --- -   - --- --- -   - -   -   - - -"
>>> encodeword "adventure"
"- ---   --- - -   - - - ---   -   --- -   ---   - - ---   - --- -   -"
```

```
>>> encodemessage "need more coffee"
"--- -   -   -   --- - -       --- ---   --- --- ---   - ---
 -   -       --- - --- -   --- --- ---   - - --- -   - - ---
 -   -   -"
>>> encodemessage "the sky is lovely"
"---   - - - -   -       - - -   --- - ---   --- - --- ---
     - -   - - -       - --- - -   --- --- ---   - - - ---
 -   - --- - -   --- - --- ---"
>>> encodemessage "live life"
"- --- - -   - -   - - - ---   -       - --- - -   - -   - -
 --- -   -"
```

DitDah.hs Code here for reference **[CODE]** **[Without Comments]**

```
dit = "-"
dah = "---"
```

```
(+++) x y = x ++ " " ++ y

a = ('a',dit+++dah)
b = ('b',dah+++dit+++dit+++dit)
c = ('c',dah+++dit+++dah+++dit)
d = ('d',dah+++dit+++dit)
e = ('e',dit)
f = ('f',dit+++dit+++dah+++dit)
g = ('g',dah+++dah+++dit)
h = ('h',dit+++dit+++dit+++dit)
i = ('i',dit+++dit)
j = ('j',dit+++dah+++dah+++dah)
k = ('k',dah+++dit+++dah)
l = ('l',dit+++dah+++dit+++dit)
m = ('m',dah+++dah)
n = ('n',dah+++dit)
o = ('o',dah+++dah+++dah)
p = ('p',dit+++dah+++dah+++dit)
q = ('q',dah+++dah+++dit+++dah)
r = ('r',dit+++dah+++dit)
s = ('s',dit+++dit+++dit)
t = ('t',dah)
u = ('u',dit+++dit+++dah)
v = ('v',dit+++dit+++dit+++dah)
w = ('w',dit+++dah+++dah)
x = ('x',dah+++dit+++dit+++dah)
y = ('y',dah+++dit+++dah+++dah)
z = ('z',dah+++dah+++dit+++dit)

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 key alist = head [ (k,v) | (k,v) <- alist, k == key ]

find letter = snd $ assoc letter symbols

addletter x y = x ++ "   " ++ y

addword x y = x ++ "       " ++ y

droplast3 w = reverse ( drop 3 ( reverse w ) )
```

```
droplast7 w = reverse ( drop 7 ( reverse w )

encodeletter x = find x -- snd ( assoc x symbols )

encodeword w = droplast3 almost
   where almost = foldr addletter "" ( map encodeletter w )

encodemessage m = droplast7 almost
   where almost = foldr addword "" ( map encodeword ( words m ) )
```