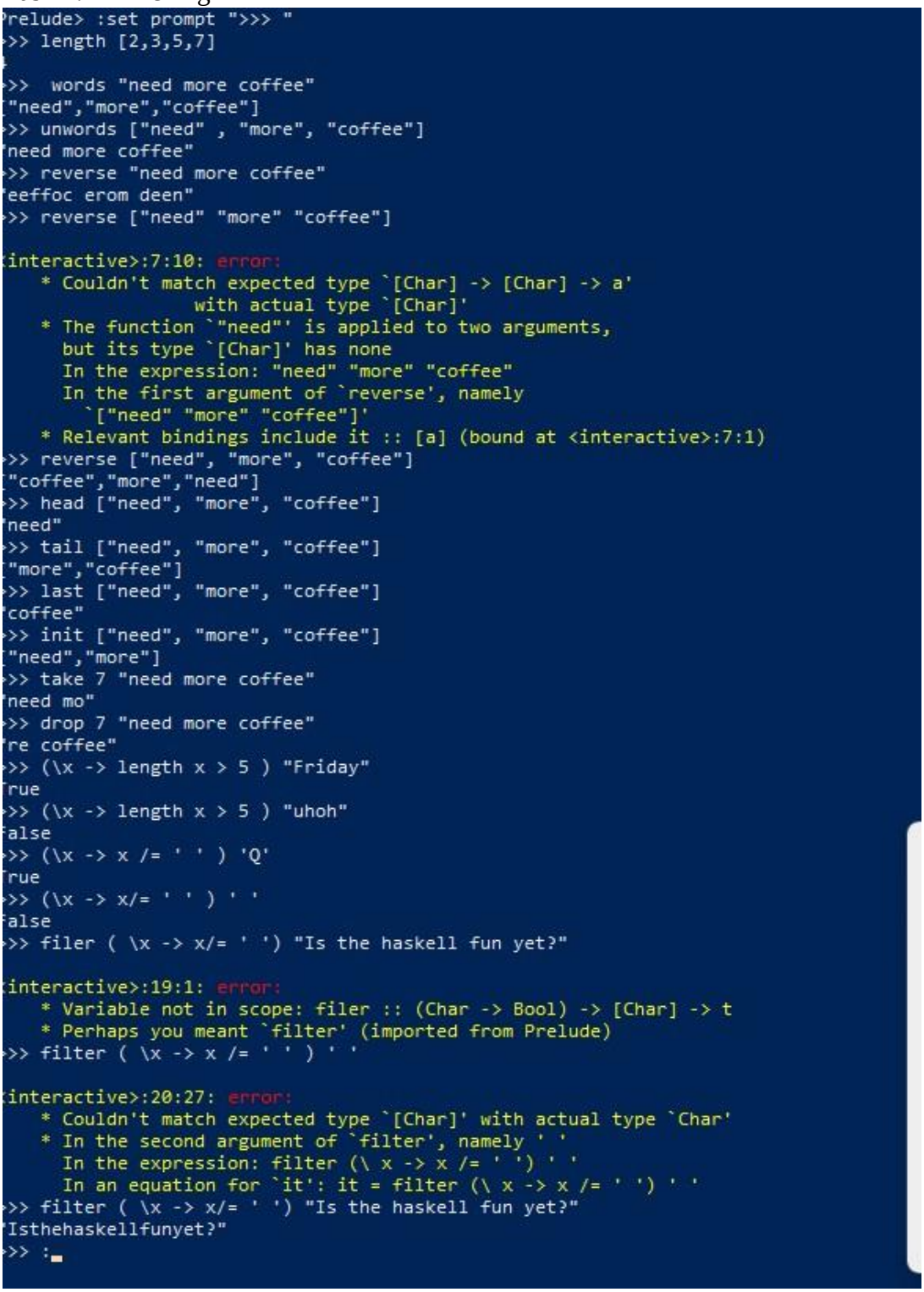
Kieran Finnegan

Haskell Assignment

Task 1: Mimicking



Task 2: Numeric Functions

Code:

```haskell
--Task 2
squareArea sideLength = sideLength * sideLength

circleArea radius = radius * radius * pi

blueAreaOfCube sideLength = blueAreaPerSide * 6
  where blueAreaPerSide = (squareArea sideLength) - (circleArea (0.25 * sideLength) )

paintedCube1 :: Double -> Double
paintedCube1 x = if (x > 2) then (6 *(x - 2)^2) else 0

paintedCube2 :: Double -> Double
paintedCube2 x = if (x > 2) then (12 * (x - 2)^2) else 0
```

Demo:

```
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
>>> map blueAreaOfCube [1..3]
[4.821902754903828,19.287611019615312,43.39712479413445]
>>> paintedCube1 1
0.0
>>> paintedCube1 2
0.0
>>> paintedCube1 3
6.0
>>> map paintedCube1 [1..10]
[0.0,0.0,6.0,24.0,54.0,96.0,150.0,216.0,294.0,384.0]
>>> paintedCube2 1
0.0
>>> paintedCube2 2
0.0
>>> paintedCube 3

<interactive>:19:1: error:
    * Variable not in scope: paintedCube :: t0 -> t
    * Perhaps you meant one of these:
        `paintedCube1' (line 11), `paintedCube2' (line 14)
>>> paintedCube2 3
12.0
>>> map paintedCube2 [1..10]
[0.0,0.0,12.0,48.0,108.0,192.0,300.0,432.0,588.0,768.0]
```

Task 3: Puzzlers

Code:

```
--Task 3
reverseWords :: String -> String
reverseWords characterString = unwords (reverse (words characterString))

averageWordLength :: Fractional a => [Char] -> a
averageWordLength characterString = fromIntegral numberCharacters / fromIntegral numberWords
  where numberCharacters = length(filter (/=' ') characterString)
        numberWords = length(words characterString)
```

Demo:

```
>>> 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
>>>
```

Task 4: Recursive List Processors

Code:

```
--Task 4
list2Set:: Eq a => [a] -> [a]
list2Set [] = []
list2set (x:xs) = if (elem x xs) then (list2set xs) else (x: list2set xs)

isPalindrome [a] = True
isPalindrome (x:xs) = if x == last xs then isPalindrome (init xs) else False
```

Demo:

Task 5: List Comprehensions

Code:

```
--Task 5
count :: Eq a => a -> [a] -> Int
count number xs = length [x | x <- xs, x == number]

freqTable :: Eq a => [a] -> [(a, Int)]
freqTable xs = [(x,y) | x <- list2Set xs, y <- [count number xs | number <- [x]]]
```

Demo:

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

Task 6: Higher Order Functions

Code:

```
--Task6
tgl :: (Num b, Enum b) => b -> b
tgl number = foldl (+) 0 [1..number]



triangleSequence :: (Num b, Enum b) => b -> [b]
triangleSequence number = map (tgl) [1..number]

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

lcsim :: (a -> b) -> (a -> Bool) -> [a] -> [b]
lcsim function predicate list = map (function) (filter (predicate) list)
```

Demo:

```
>>> tgl 5
15
>>> tgl 10
55
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> triangleSequence 10
[1,3,6,10,15,21,28,36,45,55]
>>> vowelCount "cat"
1
>>> vowelCount "mouse"
3
>>> lcsim tgl odd [1..15]
[1,6,15,28,45,66,91,120]
>>> animals = ["elephant","loin","tiger","orangatan","jaguar"]
>>> lcsim length (\w -> elem ( head w) "aeiou" ) animals
[8,9]
>>>
```

Task 7: An Interesting Statistic

Code:

```
pairwiseValues :: [Int] -> [(Int,Int)]
pairwiseValues (x:xs) = zip (x:xs) xs

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

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


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

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

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

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

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

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

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

Demo:

```
-- Test data
a :: [Int]
a = [2,5,1,3]

b :: [Int]
b = [1,3,6,2,5]

c :: [Int]
c =

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' (line 81), pairwiseHalves' (line 94)
>>> 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

<interactive>:54:1: error:
    * Variable not in scope: pairwiseDifferences :: [Int] -> t
    * Perhaps you meant `pairwiseDifference' (line 84)
>>> pairwiseDifference a
[-3,4,-2]
>>> pairwiseDifference b
[-2,-3,4,-3]
>>> pairwiseDifference c
[0,2,1,0,-1,0,-2,0,-4]
>>> pairwiseDifferences u

<interactive>:58:1: error:
    * Variable not in scope: pairwiseDifferences :: [Int] -> t
    * Perhaps you meant `pairwiseDifference' (line 84)
>>> pairwiseDifference

<interactive>:59:1: error:
    * No instance for (Show ([Int] -> [Int]))
        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
>>> pairwiseDifference u
[0,0,0,0,0,0,0,0,0]
>>> pairwiseDifference 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]
>>> pairwiseTermPairs a
[(-3,3.5),(4,3.0),(-2,2.0)]
>>> pairwiseTermsPairs b

<interactive>:76:1: error:
    * Variable not in scope: pairwiseTermsPairs :: [Int] -> t
    * Perhaps you meant one of these:
        `pairwiseTermPairs' (line 100), `pairwiseTerms' (line 106)
>>> 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 bc

<interactive>:87:15: error:
    * Variable not in scope: bc :: [Int]
    * Perhaps you meant one of these: `b' (line 69), `c' (line 72)
>>> 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
38.09523809523809
>>> nPVI c
37.03703703703703
>>> nPVI u
0.0
>>> nPVI x
124.98316498316497
>>>
```