Haskell Programming Assignment: Various Computations

Abstract: This assignment essentially introduces us to haskell and eventually delves into declaring functions, recursive function and higher order functions

---

Task 1 - Mindfully Mimicking the Demo

```
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’
```

```
>>>  ( \x -> x /= ' ' ) 'Q'
True
>>>  ( \x -> x /= ' ' ) ' '
False
```

```
>>> filter ( \x -> x /= ' ' ) "Is the Haskell fun yet?"
"IstheHaskellfunyet?"
```

Task 2 - Numeric Function Definitions
Code:

```
squareArea s = s * s

circleArea r = pi *  r^2

blueAreaOfCube s = (squareArea s - circleArea(s / 4)) * 6

paintedCube1 order = if order < 3 then 0 else 6 * (( order - 2 ) ^ 2)

paintedCube2 order = if order < 3 then 0 else 12 * ( order - 2 )
```

Demo:

```
>>> 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

Code:

```
reverseWords wordString = unwords ( reverse ( words wordString ) )
```

```
averageWordLength w = fromIntegral ( sum letterCountList ) / fromIntegral
wordC
                where wList = words w
                      wordC = length wList
                      letterCountList = map length wList
```

Demo: