EXERCISES

Q1. Would the following piece of Haskell work: 3:[True, False]? Why or why not?
Solution: No the following piece of haskell won't work because lists in haskell store homogenous data, and since 3 and True have different data types, it will result in an error

Q2. Write a function cons8 that takes a list and conses 8 on to it. Test it out on the following lists by doing:

a. cons8 []
b. cons8 [1,2,3]
c. cons8 [True,False]
d. let foo = cons8 [1,2,3]
e. cons8 foo

Solution: cons8 a = 8:a

Q3. Adapt the above function in a way that 8 is at the end of the list
Solution: cons8 a = a ++ [8]

Q4. Which of these are valid Haskell and which are not? Rewrite in cons notation.

a. [1,2,3,[]]
b. [1,[2,3],4]
c. [[1,2,3],[]]

Solution: Only c is valid. [1, 2, 3]:[]:[]

Q5. Which of these are valid Haskell, and which are not? Rewrite in comma and bracket notation.

a. []:[[1,2,3],[4,5,6]]
b. []:[]
c. []:[]:[]
d. [1]:[]:[]
e. ["hi"]:[1]:[]

Solution: a, b, c, d are valid

a. [[],[1,2,3],[4,5,6]]
b. [[]]
c. [[],[]]
d. [[1],[]]

Q6. Which of these are valid Haskell, and why?

a. 1:(2,3)
b. (2,4):(2,3)
c. (2,4):[]
d. [(2,4),(5,5),('a','b')]

e. ([2,4],[2,2])

Solution: c, e are valid. `c` because a tuple can be an element of a list, and since only one element is present, homogenous criterion is also satisfied. `e` because a tuple can contain elements of any data type.

Q7. Use a combination of fst and snd to extract the 4 from the tuple (("Hello", 4), True)

Solution: snd (fst (("Hello", 4), True))

Q8. Write a function, which returns the head and the tail of a list as the first and second elements of a tuple.

Solution: tup (x:s) = (x, s)

Q9. Use head and tail to write a function, which gives the fifth element of a list. Then, make a critique of it, pointing out any annoyances and pitfalls you can identify.

Solution: fifth a = head (tail (tail (tail (tail a))))

Q9. Define a recursive function power such that power x y raises x to the y power.

Sol:

```
pow :: Integer -> Integer -> Integer
pow x 0 = 1
pow x 1 = x
pow x n = x * (pow x (n-1))
```

Q10. Write a function isVowel :: Char -> Bool that returns True for vowels (a, e, i, o, u, as well as A, E etc.) and False for all other characters.

For example:

```
isVowel ’e’ = True
isVowel ’U’ = True
isVowel ’c’ = False
isVowel ’7’ = False
isVowel ’C’ = False
isVowel ’ ’ = False
```

Sol:

```
isVowel :: Char -> Bool
isVowel a
        | a == 'a' || a == 'e' || a == 'i' || a == 'o' || a == 'u' = True
        | otherwise = False
```

Q11. Write a function m :: String -> Int that computes the number of vowels in a string minus the number of non-vowels in the string.

For example:

```
m "" = 0
m "Amoebae Are OK" = 2
m "syzygy" = -6
m "Haskell rules!" = -6
```

```
m "cafe au lait" = 0
m "aquaria" = 3
```

Sol:

```
m :: [Char] -> Integer
m "" = 0
m (x:s) | x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u' || x == 'A' || x == 'E' || x == 'I' || x
==      'O' || x == 'U' = (m s) + 1 | otherwise = (m s) - 1
```

Q12. Write a function f :: [Int] -> [Int] that produces a list of distances between consecutive numbers in a list, in those cases where the first number is less than the second number.
For example:

```
f [4,2,5,6,1,8] = [3,1,7]
f [] = []
f [3] = []
f [3,3,1,-3] = []
```

Sol:

```
f :: [Int] -> [Int]
f [] = []
f [a] = []
f (a:b:s)
   | a < b = (b-a):f(b:s)
   | otherwise = f(b:s)
```