1. Try doing the following List comprehensions and note down the change in the outputs:
    a. [(x,y) | x <- [1,2,3], y <- [4,5]]
        i. [(1,4),(1,5),(2,4),(2,5),(3,4),(3,5)]
    b. [(x,y) | y <- [4,5], x <- [1,2,3]]
        i. [(1,4),(2,4),(3,4),(1,5),(2,5),(3,5)]
    c. [(x,y) | x <- [1..3], y <- [x..3]]
        i. [(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)]
2. Implement the following function and write down the output:

```
pairs :: [a] -> [(a,a)]
pairs xs = zip xs (tail xs)
```

    a. [(1,2),(2,3),(3,4)]
3. Which of the following are legal list constructions?

```
list1 = 1 : []
list2 = 1 : [] : []
list3 = 1 : [1]
list4 = [] : [1]
list5 = [1] : [1] : []
```

    a. list1, list3 and list5
4. Using a predicate we can define a function that maps a positive integer to its list of factors as follows:

```
factors :: Int -> [Int]
factors n = [x | x <- [1..n], n `mod` x == 0]
```

    a. By making use of this function check whether a number is prime or not

```
check_prime :: Int -> Bool
check_prime x
        | length (factors x) == 2 = True
        | otherwise = False
```

    b. Generate Prime numbers up to a limit

```
limit_primes :: Int -> [Int]
limit_primes lim = [x | x <- [1..lim], length (factors x) == 2]
```

5. **Generate all Perfect numbers up to a limit n** by making use of the above factors function

```
perfect_numbers :: Int -> [Int]
perfect_numbers n = [x | x <- [1..n], sum (init (factors x)) == x]
```

6. Write a function length' **to get length of a list.**(can use the built in function sum)

```
length' :: [a] -> Int
length' list1 = sum [1 | x <- list1]
```

7. Write a function that **takes a string and removes everything except uppercase letters** from it.

```
removeAllButUpper :: [Char] -> [Char]
removeAllButUpper list1 = [x | x <- list1, x `elem` ['A'..'Z']]
```

8. Write a function to **generate all triangles with sides equal to or smaller than 10**

gentriangle = [(a, b, c) | a <- [1..10], b <- [1..10], c <- [1..10], a + b > c, b + c > a, a + c > b]

9. Implement the following function and note down the output

```
count :: Char -> String -> Int
count x xs = length [x1 | x1 <- xs, x == x1]
```

**Main> count 'a' "ashutoshaaaa"*

5

10. Implement a function Pythagorean to **check whether a given list is a Pythagorean triple**. A triple (x,y,z) of positive integers is called Pythagorean if x2 + y2 = z2.

```
pythagorean_triplet :: (Num a, Eq a) => (a, a, a) -> Bool
pythagorean_triplet (a, b, c) | a^2 + b^2 == c^2 = True | otherwise = False
```

11. Using a list comprehension, **define a function** pyths :: Int -> [(Int,Int,Int)] **that maps an integer n to all such triples** with components in [1..n].

```
pyths :: Int -> [(Int, Int, Int)]
pyths n = [(a, b, c) | a <- [1..n], b <- [1..n], c <- [1..n], a^2 + b^2 == c^2]
```