```haskell
-- Homework 1

-- greeting = "Hello, world!"

-- Haskell Warmup
-- Question 1
largest :: String -> String -> String
largest string1 string2 = if (length string2) > (length string1) then string2 else
string1

{-
-- This solution uses guards
largest :: String -> String -> String
largest string1 string2
   | (length string2) > (length string1) = string2
   | otherwise = string1
-}


-- Question 2
-- fixed version
-- Haskell is evaluated left to right, so
-- we need to put the recursive call's parameter in parentheses
reflect :: Integer -> Integer
reflect 0 = 0
reflect num
 | num < 0 = (-1) + reflect (num+1)
 | num > 0 = 1 + reflect (num-1)


-- Question 3, part a
-- takes in type Int and returns a list of Int
all_factors :: Int -> [Int]
all_factors int = [n | n <- [1..int], mod int n == 0]

-- Question 3, part b
-- all_factors 6 returns [1, 2, 3], so we need to sum this list
-- then check if its equal to the parameter (6). if it is,
-- then its a perfect number and we add it to the perfect number's list
perfect_numbers = [x | x <- [1..], sum (init (all_factors x)) == x]
```

```haskell
-- Question 4
-- takes in 1 Integer arg and returns a Bool
-- arg will always be positive
-- Only "+", "-", and "=" are allowed

-- Logic:
-- Base Case: if number = 0, then it is even => false
-- Recursive Case: if

{-
-- Using if statements
is_odd :: Integer -> Bool
is_odd int = if int == 0
             then False
             else is_even(int - 1)

is_even :: Integer -> Bool
is_even int = if int == 0
              then True
              else is_odd(int - 1)

-- Using Guards
is_odd :: Integer -> Bool
is_odd int
 | (int == 0) = False
 | otherwise = is_even(int - 1)

is_even :: Integer -> Bool
is_even int
 | (int == 0) = True
 | otherwise = is_odd(int - 1)
-}

-- Using Pattern Matching
is_odd :: Integer -> Bool
is_odd 0 = False
is_odd int = is_even(int - 1)

is_even :: Integer -> Bool
is_even 0 = True
is_even int = is_odd(int - 1)
```

```haskell
-- Question 5
-- -- count_occurrences lst1 lst2 = if (length lst1) > (length list2) then 0
count_occurrences :: [Int] -> [Int] -> Int
count_occurrences [] _ = 1
count_occurrences _ [] = 0
count_occurrences (x:xs) (y:ys) = if x == y
                                  then count_occurrences (x:xs) ys +
count_occurrences xs ys
                                else
                                  count_occurrences (x:xs) ys
```