---------------------------------------------------------------

--

-- Starting file for Lab 9 (Spring 2017)

--

---------------------------------------------------------------

import Data.Char -- load Data.Char module to get definition of isUpper

data Shape = Circle Float

| Rect Double Double

deriving (Show,Eq)

data Device = Gizmo Int Bool

| Widget Float Char

deriving (Show,Eq)

smallEven :: Int -> Bool

smallEven x = even x && x < 10

contrived :: Int -> Char -> Bool

contrived x c = isUpper c && odd x

---------------------------------------------------------------

-- The examples from the lab writeup

---------------------------------------------------------------

--

-- Example 1: [Int]

--

-- :t ansEx2 gives the type [Int]

ansEx1 = [length "abc"]

--

-- Example 2: [Float]

--

-- Two versions:

-- :t ansEx2 gives the type [Float]

-- :t ansEx2' gives the type [Float]

ansEx2 = getList []

where

getList vs = snd (map Circle vs, vs)

ansEx2' = [x]

where

Circle x = Circle 3.21

--

-- Example 3: Bool -> Int

--

-- Two versions:

-- :t ansEx3 gives the type Bool -> Int

-- :t ansEx3' gives the type Bool -> Int

ansEx3 = sample

where

sample b

| b = length []

| otherwise = 15

ansEx3' b

| b = length []

| otherwise = 15

--

-- Example 4: [Float] -> [Shape]

--

-- :t ansEx4 gives the type [Float] -> [Shape]

ansEx4 = map Circle

---------------------------------------------------------------

-- Your problems

---------------------------------------------------------------

replaceMe = error "replace with your own code"

-- :t one should give Device

one = Gizmo 1 True

-- :t two should give (String, Int -> Bool)

two = ("two", smallEven)

-- :t three should give String -> Bool

three t

| t == "three" = True

| otherwise = False

-- :t four should give [Char] -> [Bool]

four c

| c == "four" = [True]

| otherwise = [False]

-- :t five should give [Bool -> Device]

five = [Gizmo 5]

-- :t six should give (Bool,Int,Char) -> Char

six = (\(True, 6, 'c') -> 'a')

-- :t seven should give ((Char -> Bool) -> Char) -> Bool

seven c

| isUpper c == True = True

| otherwise = False

-- :t eight should give [[Double] -> [Shape]]

eight = [map (Rect 8.0)]

-- :t nine should give Char -> Bool -> Char

nine 'a' True = 'b'

-- :t ten should give a -> b -> b

ten a b = b

-- :t eleven should give (String,[a]) -> [(Char,a)]

eleven ("abc",[a]) = [('b',a)]

-- :t twelve should give [(a,b) -> c] -> [a -> b -> c]

twelve a b c = [(\(a,b) -> c)]

-- :t thirteen should give (a -> b) -> (b -> c) -> a -> c

thirteen = replaceMe -- could not get around to this in time

-- :t fourteen should give (Char -> Char -> Bool) -> Char -> Bool

fourteen alpha c

| alpha 'a' 'b' = isUpper c

| otherwise = isLower c