Assignment 0

This first assignment asks you to write simple functions based on the types we have seen so far.

Stub methods have been created for you, you will need to fix them.

You can run the tests loading the file then doing runTests.

1. Using guards, write a function compareAges that takes as input two integer ages and returns one of three string messages: “same age”, “first is older”, or “second is older”.

2. Write a function arePythagorean which is given three integers lengths and returns true if they are all positive and if the sum of the squares of the first two equals the square of the third (these are “Pythagorean triples” because they satisfy the Pythagorean theorem equation and can form the three sides of a right-angle triangle). You may want to use guards to handle the negative cases.

3. Write functions nor1 and nor2 that both perform the NOR boolean operator. One of them should do it by listing all the individual cases. The other should do it by using the fact that “NOR” stands for “not or”.

4. Write a function getMiddle that given three numbers returns the one that is numerically in the middle. For example getMiddle 3 8 4 = 4.

```
-- Assignment 0
-- Name:

import Test.QuickCheck

compareAges :: Integer -> Integer -> String
compareAges age1 age2 = "not right"


arePythagorean :: Integer -> Integer -> Integer -> Bool
arePythagorean a b c = False

nor1 :: Bool -> Bool -> Bool
nor1 True True = True      -- fix and add more cases


nor2 :: Bool -> Bool -> Bool
nor2 x y = False           -- fix

getMiddle :: Integer -> Integer -> Integer
getMiddle a b c = b        -- fix

prop_agesWork a1 a2 =
    (a1 == a2 && result == "same age") ||
    (a1 > a2 && result == "first is older") ||
    (a1 < a2 && result == "second is older")
    where result = compareAges a1 a2
```

```
prop_norsMatch x y = nor1 x y == nor2 x y

prop_norsXX x = nor1 x x == not x
prop_norsXTrue x = not (nor1 x True) && not (nor1 True x)
prop_norsXFalse x = nor1 x False == not x

prop_getMiddle1 x =  getMiddle x (x + 3) (x - 2) == x
prop_getMiddle2 x =  getMiddle x (x - 2) (x + 3) == x
prop_getMiddle3 x =  getMiddle (x-2) x (x + 3) == x
prop_getMiddle4 x =  getMiddle (x + 3) x (x-2) == x
prop_getMiddle5 x =  getMiddle (x-2) (x + 3) x == x
prop_getMiddle6 x =  getMiddle (x + 3) (x-2) x == x


testCompareAges = do
    putStrLn "\nTesting prop_agesWork"
    putStr "prop_agesWork: "; quickCheck prop_agesWork

testPythagorean = do
    putStrLn "\nTesting arePythagorean"
    testFail "negative a" $ arePythagorean (-3) 4 5
    testFail "negative b" $ arePythagorean 3 (-4) 5
    testFail "negative c" $ arePythagorean 3 4 (-5)
    test "valid" $ arePythagorean 3 4 5
    testFail "invalid" $ arePythagorean 3 4 6

testNors = do
    putStrLn "\nTesting nor1 and nor2"
    putStr "prop_norsXX: "; quickCheck prop_norsXX
    putStr "prop_norsXTrue: "; quickCheck prop_norsXTrue
    putStr "prop_norsXFalse: "; quickCheck prop_norsXFalse

testGetMiddle = do
    putStrLn "\nTesting getMiddle"
    putStr "prop_getMiddle1: "; quickCheck prop_getMiddle1
    putStr "prop_getMiddle2: "; quickCheck prop_getMiddle2
    putStr "prop_getMiddle3: "; quickCheck prop_getMiddle3
    putStr "prop_getMiddle4: "; quickCheck prop_getMiddle4
    putStr "prop_getMiddle5: "; quickCheck prop_getMiddle5
    putStr "prop_getMiddle6: "; quickCheck prop_getMiddle6

test s b = do
    putStr s
    putStr ": "
    putStrLn $ if b then "OK" else "FAILED"

testFail s b = test s (not b)

runTests :: IO ()
runTests = do
    testCompareAges
    testPythagorean
    testNors
    testGetMiddle
```