Тестирование

Haskell

Упражнение

- Напишите функцию, которая считает частоту каждого элемента в списке:

```
frequencies :: Eq a => [a] -> [(a,Int)]
frequencies = undefined
```

Тестирование

Подходы

- тестовый пример
- свойство

Библиотеки

- hspec
- QuickCheck
- hedgehog
- doctest

hspec

```
import Test.Hspec

main :: IO ()
main = hspec $ do
  describe "Prelude.head" $ do
    it "returns the first element of a list" $ do
      head [23 ..] `shouldBe` (23 :: Int)
```

https://hspec.github.io/

Упражнение

- Придумайте и напишите два тестовых примера для функции frequencies

QuickCheck

```
import Test.QuickCheck

prop_commutativeAdd :: Int -> Int -> Property
prop_commutativeAdd x y = x + y === y + x

ghci>quickCheck prop_commutativeAdd
+++ OK, passed 100 tests.
```

QuickCheck

```
prop_length1 xs = length (tail xs) == (length xs - 1)


ghci> quickCheck prop_length1
*** Failed! Exception: 'Prelude.tail: empty list' (after 1 test):
[]
```

(==>)

```
prop_length2 xs = not (null xs) ==> length (tail xs) == (length xs - 1)

ghci> quickCheck prop_length2
+++ OK, passed 100 tests; 15 discarded.

ghci> verboseCheck prop_length2
Skipped (precondition false):
[]
Passed:
[()]
Passed:
[(),(),(),(),(),()]
Passed:
[(),(),(),(),(),(),(),()]
...
```

NonEmptyList, (===)

```
prop_length3 :: NonEmptyList Int -> Property
prop_length3 (NonEmpty xs) = length (tail xs) === length xs - 1
```

```
ghci> verboseCheck prop_length3
Passed:
NonEmpty {getNonEmpty = [0]}
0 == 0
Passed:
NonEmpty {getNonEmpty = [1,2]}
1 == 1
Passed:
NonEmpty {getNonEmpty = [-1,-2]}
1 == 1
```

Упражнение

Запишите свойства функции rev

- Дважды перевернутый список === исходный список
- Перевернутая конкатенация двух списков === конкатенация двух перевернутых списков

```
rev :: [a] -> [a]
rev [] = []
rev (x:xs) = xs ++ [x]
```

ошибка

```
import Data.Char

propToUpperChanges :: Char -> Property
propToUpperChanges c = toUpper c =/= c

ghci> quickCheck propToUpperChanges
*** Failed! Falsified (after 2 tests and 1 shrink):
'A'
'A' == 'A'
```

forAll

```
import Data.Char

propToUpperChanges :: Char -> Property
propToUpperChanges c = toUpper c =/= c

propToUpperChangesLetter :: Property
propToUpperChangesLetter = forAll (elements ['a'..'z']) propToUpperChanges

ghci> quickCheck propToUpperChangesLetter
+++ OK, passed 100 tests.
```

Упражнение (frequencies)

Проверьте, что сумма частот элементов равняется длине исходного списка. Для этого напишите функцию, первый аргумент которой – функция для проверки. Протестируйте свою реализацию frequencies. Убедитесь, что функция freq1 удовлетворяет этому свойству.

```
sumIsLength :: ([Char] -> [(Char,Int)]) -> [Char] ->  Property
sumIsLength freq input = undefined

freq1 :: Eq a => [a] -> [(a,Int)]
freq1 [] = []
freq1 [x] = [(x,1)]
freq1 (x:y:xs) = [(x,1),(y,length xs + 1)]

ghci> quickCheck $ sumIsLength freq1
+++ OK, passed 100 tests.
```

Упражнение (frequencies)

Проверьте свойство: каждый элемент из исходного списка должен иметь какую-то частоту в ответе.
Подсказка: используйте forAll

```
inputInOutput :: ([Char] -> [(Char,Int)]) ->
                                NonEmptyList Char -> Property
inputInOutput freq (NonEmpty input) = undefined

freq2 :: Eq a => [a] -> [(a,Int)]
freq2 xs = map (\x -> (x,1)) xs
```

Упражнение (frequencies)

Проверьте свойство: для любой пары (x,n) из ответа x встречается в исходном списке ровно n раз

```
outputInInput :: ([Char] -> [(Char,Int)]) ->
                 NonEmptyList Char -> Property
outputInInput freq (NonEmpty input) = undefined

freq3 :: Eq a => [a] -> [(a,Int)]
freq3 [] = []
freq3 (x:xs) = [(x,1 + length (filter (==x) xs))]

ghci> quickCheck $ outputInInput freq3
+++ OK, passed 100 tests.
```

QuickCheck + hspec

```
import Test.Hspec
import Test.QuickCheck

main :: IO ()
main = hspec $ do
  describe "Prelude.head" $ do
    it "returns the first element of a list" $ do
      head [23 ..] `shouldBe` (23 :: Int)

    it "returns the first element of an *arbitrary* list" $
      property $ \x xs -> head (x:xs) == (x :: Int)
```

within

```
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

$$(n+1)!! = \frac{(n+1)!}{n!!}$$

```
doubleFactorial 0 = 1
doubleFactorial n = n * doubleFactorial (n - 2)

prop_factorial :: NonNegative Integer -> Property
prop_factorial (NonNegative n) = within 1000000 $
      doubleFactorial (n + 1) === factorial (n + 1) `div` doubleFactorial n

ghci> quickCheck prop_factorial
*** Failed! Timeout of 1000000 microseconds exceeded. (after 1 test):
NonNegative {getNonNegative = 0}
```

Debug.Trace

```
import Debug.Trace

doubleFactorial 0 = 1
doubleFactorial n = trace ("doubleFactorial " ++ show n) $
                              n * doubleFactorial (n - 2)

ghci> doubleFactorial 5
doubleFactorial 5
doubleFactorial 3
doubleFactorial 1
doubleFactorial -1
doubleFactorial -3
doubleFactorial -5
doubleFactorial -7
doubleFactorial -9
```

Создание проекта с cabal

- Создаем директорию для проекта

>mkdir myfirstapp

> cd myfirstapp

- Создаем новый cabal-проект

>cabal init

- Запускаем приложение

> cabal run myfirstapp

https://cabal.readthedocs.io/en/3.6/getting-started.html#creating-a-new-application

Создание проекта со stack

- Создаем новый проект

>stack new mysecondapp

- Переходим в созданную директорию и собираем проект

> cd mysecondapp

> stack build

- Запускаем исполняемый файл

>stack exec mysecondapp-exe

https://docs.haskellstack.org/en/stable/GUIDE/