

Haskell演習問題 解答

2012年9月4日

村田 裕哉

解答1

murata1.hs

```
includedTax :: Int -> Int -> Int
```

```
includedTax rate value = value + consumptionTax rate value
```

```
consumptionTax :: Int -> Int -> Int
```

```
consumptionTax rate value = value * rate `div` 100
```

除算の演算子は色々ある.

/ とか div とか quot とか.

使い方や, 結果が異なる場合がある.

どの除算関数を使えば良い？

- 「/」は、実数の除算で用いる.

例: `ghci> 5 / (-2)`
-2.5

- 整数の除算には2組の関数がある.

`quotRem x y = (q,r)`

rの符号はxと同じ.

`divMod x y = (d,m)`

mの符号はyと同じ.

解答2

murata2.hs

```
includedTax :: Int -> Int -> Int
```

```
includedTax value year = value + consumptionTax value year
```

```
consumptionTax :: Int -> Int -> Int
```

```
consumptionTax value year = value * taxRate year `div` 100
```

```
taxRate :: Int -> Int
```

```
taxRate year = if year < 1989 then 0
```

```
                else if year < 1997 then 3
```

```
                else if year < 2014 then 5
```

```
                else if year == 2014 then 8
```

```
                else 10
```

解答3

murata3.hs

```
cashRegister :: [Int] -> Int -> Int
```

```
cashRegister valueList year = includedTax (sum valueList) year
```

Sum valueList の部分のカッコは必須.

吉井 英人

解答1

`[x | x <- [1..100], even x, x `mod` 7 == 0]`

解答2

```
let rightTriangles' =  
  [(a,b,c) | c <- [1..10], b <- [1..c],  
    a <- [1..b], a^2 + b^2 == c^2,  
    a+b+c == 12]
```

解答3

fibonacci.hs

```
fibonacci :: (Integral a) => a -> a
```

```
fibonacci 0 = 0
```

```
fibonacci 1 = 1
```

```
fibonacci n = fibonacci (n-2) + fibonacci(n-1)
```

その他フィボナッチについて以下のURLを参照

http://www.haskell.org/haskellwiki/The_Fibonacci_sequence

葛迫 祐介

解答1

```
fizzbuzz :: (Integral a) => a -> String
```

```
fizzbuzz x
```

```
  | x `mod` 15 == 0 = "FizzBuzz"
```

```
  | x `mod` 3  == 0 = "Fizz"
```

```
  | x `mod` 5  == 0 = "Buzz"
```

```
  | otherwise = show x
```

```
Fizzbuzz' :: (Integral a) => [a] -> [a]
```

```
fizzbuzz' list = map fizzbuzz list
```

※本解答では、リストを引数にとっている。

解答2

```
fib :: (Integral a) => a -> a
```

```
fib 0 = 0
```

```
fib 1 = 1
```

```
fib n = fib (n - 1) + fib (n - 2)
```

```
Fibs :: (Integral a) => [a] -> [a]
```

```
fibs list = map fib list
```

※本解答では, リストを引数にとっている. (例:[0..n])

木村 有祐

Haskell演習問題 解答1

問. 任意の無限リストをパラメータとしてとり, このリストの各要素を15倍した要素のうち, 500以上となる要素を最小のものから10個とる関数を作成せよ.

なお, **filter**関数と**map**関数を使用すること.

答.

```
let myfil xs = take 10 (filter (>500) (map (*15) xs))  
  
myfil [1..]  
> [510,525,540,555,570,585,600,615,630,645]
```

Haskell演習問題 解答2

問. 任意の有限リストをパラメータとしてとり, このリストの各要素を7倍し3を足す関数を作成せよ.
なお, ラムダ関数を使用すること.

答.

```
let mylamda xs = map ( \a -> a*7+3) xs  
  
mylamda [1,2,3,4,5]  
> [10,17,24,31,38]
```


Haskell演習問題

解答3

問. 要素が1桁の整数である任意の有限リストをパラメータとしてとり, このリストの要素を平坦化しひとつの数値にする関数を作成せよ.

なお, **foldl**もしくは**foldr**関数を使用すること.

答.

```
let myflat xs = foldl ( \m x-> m*10 + x) 0 xs
```

```
myflat [1,2,3,4,5,6,7,8,9]
```

```
> 123456789
```

北川 初音

解答1

$f(x) = \sum_{f \in \text{fst}(x)} f$

解答2

```
f x = map digitToInt $ fst $ partition (isHexDigit) x
```

解答3

valの値が1000を超えなかった場合, head (dropWhile (¥(val,y,m,d) -> val < 1000) stock)を実行すると, dropWhileが空リストを返す. head([])はエラーを返すため, head (dropWhile (¥(val,y,m,d) -> val < 1000) stock)はエラーを返す. 一方, find (¥(val,y,m,d) -> val < 1000) stockは返り値の型がMaybeであるため, Nothingを返す. このため, find (¥(val,y,m,d) -> val < 1000) stockの方が安全に実行できる.

池田 騰

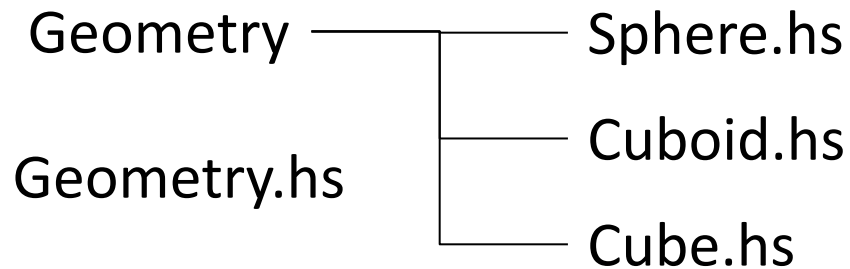
解答1

MoneyMap.hs

```
import qualified Data.Map as Map
moneyList = [("Nobita", 300), ("Doraemon",100),
             ("Suneo",5000), ("Takeshi",200), ("Shizuka",2000)]
moneyMap = Map.fromList moneyList
updatedMoneyMap = Map.insertWith (+) "Takeshi" (-100) $ Map.insertWith (+) "Nobita" (-100)
                        $ Map.insertWith (+) "Doraemon" 100 $ Map.insertWith (+) "Suneo" 2000
                        $ Map.insertWith (+) "Shizuka" 500 $ moneyMap
```

- (1) Data.Map を import する
- (2) 連想リストとして初期状態のキーと値を格納する.
- (3) Map.fromList を用いて連想リストをMapに変換する.
- (4) Map.insertWith を用いてMap の各キーに対応する値を更新する.

解答2(1/5)



上記のディレクトリ構成でファイルを作成

解答2(2/5)

Geometry.hs

```
import qualified Geometry.Sphere as Sphere
import qualified Geometry.Cuboid as Cuboid
import qualified Geometry.Cube as Cube
```

サブモジュール(Geometry.Sphere, Geometry.Cuboid, Geometry.Cube)をそれぞれ修飾インポートするファイルを作成.

各サブモジュールの定義する関数を使用したい場合, このファイルをターミナルで :l コマンド用いて以下のように読み込めば良い.

:l Geometry.hs

解答2(3/5)

Sphere.hs

```
module Geometry.Sphere
( volume,
  area
) where
volume :: Float -> Float
volume radius = (4.0 / 3.0) * pi * (radius ^ 3)
area :: Float -> Float
area radius = 4 * pi * (radius ^ 2)
```

球体の体積と表面積を求めるサブモジュールを定義するファイルを作成.

解答2(4/5)

Cuboid.hs

```
module Geometry.Cuboid
( volume,
  area
) where
volume :: Float -> Float -> Float -> Float
volume a b c = rectangleArea a b * c
area :: Float -> Float -> Float -> Float
area a b c = rectangleArea a b * 2 + rectangleArea a c * 2 + rectangleArea c b * 2
rectangleArea :: Float -> Float -> Float
rectangleArea a b = a * b
```

直方体の体積と表面積を求めるサブモジュールを定義するファイルを作成.

解答2(5/5)

Cube.hs

```
module Geometry.Cube
( volume,
  area
) where
import qualified Geometry.Cuboid as Cuboid
volume :: Float -> Float
volume side = Cuboid.volume side side side
area :: Float -> Float
area side = Cuboid.area side side side
```

立体の体積と表面積を求めるサブモジュールを定義するファイルを作成.

解答3

MoneyMap.hs

```
import qualified Data.Map as Map
moneyList = [("Nobita", 300), ("Doraemon",100),
             ("Suneo",5000), ("Takeshi",200), ("Shizuka",2000)]
moneyMap = Map.fromList moneyList
updatedMoneyMap = Map.insertWith (+) "Takeshi" (-100) $ Map.insertWith (+) "Nobita" (-100)
                      $ Map.insertWith (+) "Doraemon" 100 $ Map.insertWith (+) "Suneo" 2000
                      $ Map.insertWith (+) "Shizuka" 500 $ moneyMap
```

- (1) Data.Map を import する
- (2) 連想リストとして初期状態のキーと値を格納する.
- (3) Map.fromList を用いて連想リストをMapに変換する.
- (4) Map.insertWith を用いてMap の各キーに対応する値を更新する.

解答3

```
data PC = PC {os :: String, cpu :: String, memory :: String, year :: Int}  
           deriving (Show)
```

檀上 未来

問1

(1) Show (2) Name

問2

- (1) Right “Yoshii”
- (2) Left “doc number 5005 is not exist.”

乃村 先生

問題1

YesNo 型クラスにおいて, リストの Noは, 空の場合でしたが, これを変更して, 空リストのリストもNoとなるようにしなさい.

[a] の a も YesNo じゃないとね

```
class YesNo a where  
  yesno :: a -> Bool
```

```
instance YesNo Int where  
  yesno 0 = False  
  yesno _ = True
```

```
instance YesNo a => YesNo [a] where  
  yesno []      = False  
  yesno (x:xs) = (yesno x) || (yesno xs)
```

```
main = do  
  print $ yesno [[] :: [Int], [] :: [Int]]  
  print $ yesno [1 :: Int]
```

問題2

どういうわけで $\text{Map } k$ が Functor の一員になるのかについて説明しなさい.

Functor になるには

- Functor になるためには, fmap が必要.
- Data.Map のソースコードを読む

Map k の instance 宣言

```
instance Functor (Map k) where  
  fmap f m = map f m
```

```
map :: (a -> b) -> Map k a -> Map k b  
map f = mapWithKey (¥_ x -> f x)
```

```
mapWithKey :: (k -> a -> b) -> Map k a -> Map k b  
mapWithKey f = go
```

```
  where
```

```
    go Tip = Tip
```

```
    go (Bin sx kx x l r) = Bin sx kx (f kx x) (go l) (go r)
```

fmap: 全ての value に関数を適用して置き換える関数

問題3

doの中にI/Oを書くことで、何故安全に副作用のある関数とそうでない関数を分離できるのか。例えば、純粋な関数が戻り値を返す前に結果を画面に表示してみたい場合を考えてみよ。

Haskell でも printfデバッグは可能だが...

```
import Data.Char
import Text.Printf

readInt :: IO Int
readInt = do
  str <- getLine
  let num = read (findDigit str) :: Int
  printf "readInt: %d¥n" num
  return num
  where findDigit = filter isDigit

main = do
  n <- readInt
  print n
```

IO が副作用の目印になる

Haskell では、関数は1つの式なので

- 1) 式全体が1つの型を持つので先頭に do を置く以外 IO 不可
- 2) つまり、純粋関数の途中に do を混ぜられない
- 3) do (IO) を使った時点でその関数の型に IO が付く

この原理で、IO と純粋関数が分離できる.

例のように、全部に IO を付ければ、普通の手続き型っぽく同じことが書けるが、副作用のある関数が多くなるのは望ましくない.

Haskell の場合は、関数から ``IO`` をできるだけ減らすということが目に見えるので、より安全になる.

IO を局所化することが大事

```
import Data.Char
import Text.Printf

readInt :: String -> Int
readInt str = read (findDigit str) :: Int
  where findDigit = filter isDigit

main = do
  n <- getLine
  let num = readInt n
  printf "readInt: %d¥n" num
  print $ num
```

左海 裕庸

問題1 解答

(1) 初項 a , 公比 r の等比数列(geometric progression)のリストを作成する関数.

```
geomProg :: Num -> Num -> Num
geomProg a r = a : geomProg (a * r) r
```

(2), (3) の前にfact関数を定義.

```
fact :: Integer -> Integer
fact n = product [1..n]
```

(2) 異なる n 個から異なる m 個を選ぶ順列(permutation)の総数を求める関数.

```
perm :: Integer -> Integer -> Integer
perm n m
  | n > 0, m >= 0, n >= m = div (fact n) (fact (n - m))
  | otherwise = error "Input error:perm n m -> n > 0 & m > 0 & n > m"
```

(3) 異なる n 個から異なる m 個を選ぶ組み合わせ(combination)の総数を求める関数.

```
comb :: Integer -> Integer -> Integer
comb n m
  | n > 0, m >= 0, n >= m = div (fact n) (fact m * fact (n - m))
  | otherwise = error "Input error:comb n m -> n > 0 & m > 0 & n > m"
```

問題2 解答

エラトステネスのふるいを作成.

```
primes = sieve [2..]  
sieve :: [Integer] -> [Integer]  
sieve (p:ps) = p:sieve[n | n <- ps, mod n p /= 0]
```

確認方法例

```
ghci> take 5 primes  
[2, 3, 5, 7, 11]
```

$[n | n <- ps, \text{mod } n \ p \neq 0]$ により, 与えられた数列から p の倍数を除いた数列を得る. これを繰り返すことにより, 素数の数列が得られる.

仲尾 和祥

問題1

最大公約数

greatCommonDivisor x y =

if y == 0 then x

else if x < y then greatCommonDivisor y x

else greatCommonDivisor (x - y) y

最小公倍数

leastCommonMultiple x y =

if x <= 0 || y <= 0 then 0

else (x * y) `div` greatCommonDivisor x y

問題2

(1) [1,2,3,5,6,10]

(2) 与えられたリストを昇順にソートする関数

(3)

`cubeList [] = []`

`cubeList (x:xs) = x*x*x : cubeList xs`

`cubeSort = cubeList . foo`

深井貴明

問1の回答


`fukai1 = foldr (-) 0`

$$\begin{aligned} & x_1 - x_2 + x_3 \cdots + (-1)^{n-1} x_{n-1} + (-1)^n x_n \\ &= x_1 - (x_2 - (x_3 \cdots - (x_{n-1} - (x_n - 0)) \cdots)) \end{aligned}$$


問2の回答

fukai2 pre xs =

(length xs `div` 2) < (length \$ filter pre xs)



条件を満たす要素の
数を求める



条件を満たす要素のみの
リストを求める

福田

問1の回答

```
insert :: Ord k => k -> v -> Tree k v -> Tree k v
insert k v Empty = singleton k v
insert k v (Node xk xv l r)
  | k < xk    = Node xk xv (insert k v l) r
  | otherwise = Node xk xv l (insert k v r)
```

問2の回答

```
searchTree :: Ord k => k -> Tree k v -> Maybe v
searchTree _ Empty = Nothing
searchTree k (Node xk xv l r)
  | k == xk = Just xv
  | k < xk = searchTree k l
  | otherwise = searchTree k r
```

宮崎

Haskell練習問題(宮崎) 解答

<問題1>

与えられたリストの末尾要素を返す関数last' を定義せよ.

<解答1>

```
last' :: [a] -> a
```

```
last' [] = error "empty list"
```

```
last' [x] = x
```

```
last' (x:xs) = last' xs
```

Haskell練習問題(宮崎) 解答

<問題2>

リストの与えられた位置に1つの要素を追加する関数を定義せよ.

<解答2>

```
insertNth :: a -> Int -> [a] -> [a]
```

```
insertNth elem _ [] = [elem]
```

```
insertNth elem 0 xs = (elem:xs)
```

```
insertNth elem n (x:xs) = x:insertNth elem (n-1) xs
```

Haskell練習問題(宮崎) 解答

<問題3>

与えられた整数のリストから3 の倍数を取り出し, それらの平方の和を返す関数を定義せよ.

<解答3>

```
miyazaki3 :: (Integral a) => [a] -> a
miyazaki3 = foldr (+) 0 . map (^2) ¥
              . filter (¥x->(mod x 3)==0)
```