すごいHaskellたのしく学ぼう！読書会#3

森本達也

演習問題

演習課題解答

- パターンマッチを使ったhead'の作成
- うるう年判定プログラム（Haskell, Ruby）

パターンマッチを使ったhead'

- リストの先頭部分を返すhead'関数

```
--------
head' :: [a] -> a
head' [] = error "Empty!"
head' (x:xs) = x
--------
```

うるう年判定

- うるう年の定義

  0年からを対象に、例外を除き4で割り切れる年

  【例外】
  - 100で割り切れる年はうるう年ではない
  - 400で割り切れる年はうるう年である

うるう年判定

- コード

```
--------
leap_year :: Int -> String
leap_year leap
 | leap == 0 = "Leap!"
 | (mod leap 400) == 0 = "Leap!"
 | (mod leap 100) == 0 = "No Leap!"
 | (mod leap 4) == 0 = "leap!"
 | otherwise = "No Leap!"
--------
```

4章　Hello再帰！

第4章　Hello再帰！

- 再帰とは
- 最高に最高！
- さらにいくつかの再帰関数
  - replicate, take, reverse, repeat, zip, elem
- クイック、ソート！
  - アルゴリズム, コード

再帰

関数の定義の中で、自分自身を呼び出す事！

- 命令型言語は計算を**どうやって**するかを指定
- Haskellでは求めるものが**何であるか**を宣言

- Haskellの目的は欲しい結果が何であるかを直接定義することである。

  ⇒Haskell上では再帰が重要！

最高に最高！

- 再帰を用いてmaximum関数を定義！

- 再帰を使う際に最初に考えるべきこと
  「命令的に定義すればどうなるかを考える」
  「基底部を定義する」
  「自分自身（再帰）の呼び出し方」

maximum関数のコード

リストから、最大の数字を取り出す

```
--------
maximum' :: (Ord a) => [a] -> a
maximum' [] = error "maximum of empty list"
maximum' [x] = x
maximum'(x:xs) = max x(maximum' xs)
--------
```

さらにいくつかの再帰関数①

- Replicate　(複製)

  - 値を指定された数だけ繰り返したリストを返す

- Take (取得)

  - リストから指定された数の要素を返す

- Reverse (反対)

  - リストから、同じ要素の逆順リストを返す

さらにいくつかの再帰関数②

- Repeat　(反復)

  - 要素から無限リストを返す

- Zip (締める)

  - 2つのリストから、綴じ合わせを返す

- Elem

  - 値とリストをとり、値がリストに含まれているかを返す

Replicate

値を指定された数だけ繰り返したリストを返す

```
--------
replicate' :: Int -> a -> [a]
replicate' n x
  |n <= 0 = []
  |otherwise = x : replicate' (n-1) x
--------
```

Take

リストから指定された数の要素を返す

```
--------
take' :: Int -> [a] -> [a]
take' n _
  |n <= 0 = []
take' _ [] = []
take' n(x:xs) = x : take' (n-1) xs
--------
```

Reverse

リストから、同じ要素の逆順リストを返す

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

repeat

要素から無限リストを返す

```
--------
repeat' :: a -> [a]
repeat' x = x : repeat' x
--------
```

zip

2つのリストから、綴じ合わせを返す

```
--------
zip' :: [a] -> [b] -> [(a, b)]
zip' _ [] = []
zip' [] _ = []
zip' (x:xs) (y:ys) = (x,y) : zip' xs ys
--------
```

elem

値とリストをとり、値がリストに含まれているかを返す

```
--------
elem' :: (Eq a) => a -> [a] -> Bool
elem' a [] = False
elem' a (x:xs)
  | a == x = True
  | otherwise = a `elem'` xs]
--------
```

クイック、ソート！

- アルゴリズム　　[1, 4, 3], [9, 6, 7]
- ソートしたいリストを [5, 1, 9, 4, 6, 7, 3]とする
- 5をピボットとして選択し、[1, 4, 3], [9, 6, 7]でサンドイッチする。さらに、同じ処理を繰り返していく

[1, 4, 3], [5], [9, 6, 7]

↓

[], [1], [4, 3] , [5], [6, 7], [9]

↓

[], [1], [3], [4], [], [5], [], [6], [7], [9]

quick

```
--------
quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x :xs) =
  let smallerOrEqual = [a | a <- xs, a <= x]
      larger = [a | a <- xs, a > x]
  in quicksort smallerOrEqual ++ [x] ++ quicksort larger
--------
```