Haskell 2

Fonksiyonlar ve Özyineleme

Şevket Umut ÇAKIR

PAÜ

31 Mart 2020

Anahat

Örüntü Eşleme(Pattern Matching)

- Fonksiyon tanımında farkı örüntüler için farklı satırlar kullanılabilir
- Bu yol basit ve okunabilir kod yazmayı sağlar
- Örüntüler sayılar, karakterler, listeler, çokuzlular vd. gibi veri türlerinden olabilir

Örüntü Eşleme Örnek

```
sansli 7 = "ŞANSLI SAYI 7!"
sansli x = "Üzgünüm, şansınız yok!"

soyle 1 = "Bir"
soyle 2 = "İki"
soyle 3 = "Üç"
soyle 4 = "Dört"
soyle x = "Bir ile Dört arasında değil"

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

Örüntü Eşleme Örnek Kullanım

```
GHCi, version 8.6.3: http://www.haskell.org/ghc/  :? for
 ↪  help
Prelude> :l pattern1.hs
[1 of 1] Compiling Main             ( pattern1.hs,
 ↪  interpreted )
Ok, one module loaded.
*Main> putStrLn (sansli 7)
ŞANSLI SAYI 7!
*Main> putStrLn (sansli 4)
Üzgünüm, şansınız yok!
*Main> putStrLn (soyle 3)
Üç
*Main> faktoryel 20
2432902008176640000
```

Örüntü Eşleme

Listelerde ve Çokuzluluarda Kullanımı

- Listelerde baş ve kuyruğu ayırmak için **:** sembolü parantez ile birlikte kullanılır
- Çokzululardа elemanlar parantez içinde **,** sembolü ile birbirinden ayrılır
- Listelerde örüntü eşleme için ++ sembolü kullanılmaz

Örüntü Eşleme Örnek

```haskell
vektorTopla a b = (fst a + fst b, snd a + snd b)

vektorTopla2 (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)

soyle [] = "Liste boş"
soyle (x:[]) = "Listenin bir elemanı var: " ++ show x
soyle (x:y:[]) = "Listenin iki elemanı var: " ++ show x ++ " ve " ++ show y
soyle (x:y:_) = "Liste uzun. İlk iki eleman: " ++ show x ++ " ve " ++ show y

toplam [] = 0
toplam (x:xs) = x + toplam xs

ilkharf tumu@(x:xs) = tumu ++ " kelimesinin ilk harfi " ++ [x] ++ " dir"
```

Muhafızlar(Guards)

- Örüntü eşleme bir değerin bir biçime uygun olduğu doğrular
- Muhafızlar ise değerlerin bazı özelliklerinin doğru veya yanlışlığını test eder
- **if** yapısına benzerlik gösterir fakat daha okunur özelliktedir

Muhafız Örnek

Vücut Kitle İndeksi

Vücut kitle indeksi (VKİ), insanların kilolarını sınıflandırmak için kullanılan bir ölçüttür ve vücut kütlesinin (kg), uzunluğunun metre cinsinden karesine bölünmesiyle hesaplanır.

```
vkiSoyle vki
  | vki <= 18.5 = "Çok zayıfsınız"
  | vki <= 25 = "Normal kilodasınız"
  | vki <= 30 = "Kilolusunuz, biraz dikkat!"
  | otherwise = "Tebrikler!"
```

Muhafız Örnek

```haskell
vkiSoyle2 kilo boy
  | kilo / boy ^ 2 <= 18.5 = "Çok zayıfsınız"
  | kilo / boy ^ 2 <= 25 = "Normal kilodasınız"
  | kilo / boy ^ 2 <= 30 = "Kilolusunuz, biraz dikkat!"
  | otherwise = "Tebrikler!"

vkiSoyle3 kilo boy
  | vki <= zayif = "Çok zayıfsınız"
  | vki <= normal = "Normal kilodasınız"
  | vki <= sisman = "Kilolusunuz, biraz dikkat!"
  | otherwise = "Tebrikler!"
  where vki = kilo / boy ^ 2
        zayif = 18.5
        normal = 25.0
        sisman = 30.0
--      (zayif, kilolu, sisman) = (18.5, 25.0, 30.0) --şeklinde de
↪  olabilirdi
--kullanımı: vkiSoyle3 80 1.75
vkiHesapla l = [vki k b | (k, b) <- l]
  where vki k b = k / b ^ 2
```

Let İfadesi

- `where` ifadesine benzer olan, yerel bir kapsamda ifade veya değerleri isimlere bağlamaya yarayan bir ifadedir
- `let baglamalar in ifade` şeklinde tanımlanır
- `let` içinde örüntü eşleme kullanılabilir

Let İfadesi Örnek

```haskell
silindir r h =
  let yanAlan = 2 * pi * r *h
      ustAlan = pi * r *r
  in yanAlan + 2 * ustAlan

vkiHesapla l = [vki | (k, b) <- l, let vki = k / b ^ 2]
```

Let İfadesi Örnek

İnteraktif Yorumlayıcıda Kullanımı

```
Prelude> 4 * (let a = 9 in a + 1) + 2
42
Prelude> [let square x = x * x in (square 5, square 3,
↪  square 2)]
[(25,9,4)]
Prelude> let zoot x y z = x * y + z
Prelude> zoot 3 9 2
29
```

Case İfadesi

- Diğer dillerdeki `case` ifadelerine benzerdir. Birden fazla seçenek içinden seçimi sağlar.

```
listeTanimla x = "Liste " ++ case x of []  -> "boş."
                                       [e] -> "tek elemanlı."
                                       x -> "birden çok elemanlı."
```

Özyineleme

- Özyineleme, gövdesi içinde, bir fonksiyonun kendisini çağırmasıdır
- Meşhur Fibonacci serisinin tanımı özyinelemeli olarak verilmiştir

$$\begin{aligned} F(0) &= 1 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned} \tag{1}$$

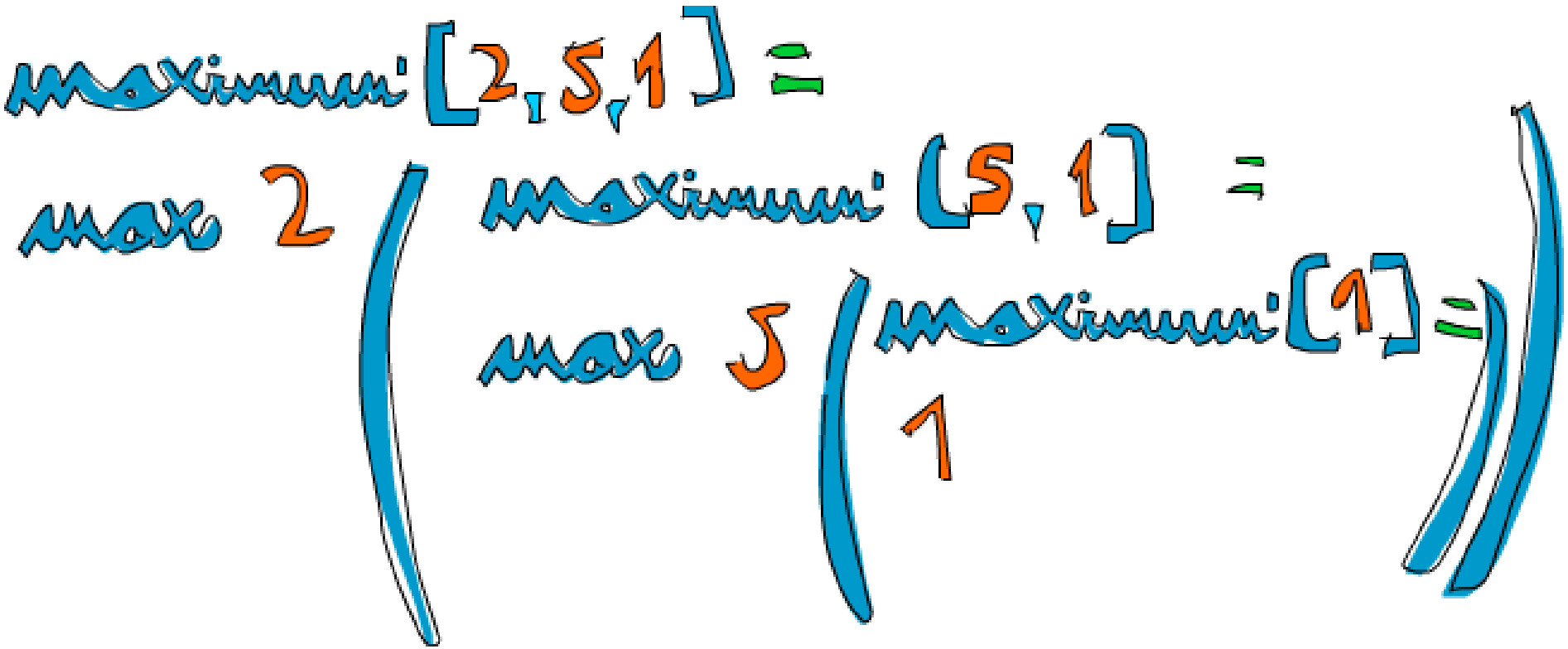
- Fonksiyon tanımında özyinelemeli olmayan kısımlara **durma noktası(edge condition)** adı verilir
- Özyinelemeli fonksiyonlar tasarlanırken durma noktasına dikkat edilmelidir
- Listelerde özyineleme genellikle kuyruğun özyinelemeli çağrıya gönderilmesi ve dönen sonucun baş ile birleştirilmesi şeklinde gerçekleşir

Özyineleme Örnek

```
maximum' [] = error "boş listenin en büyüğü"
maximum' [x] = x
maximum' (x:xs)
    | x > maxKuyruk = x
    | otherwise = maxKuyruk
    where maxTail = maximum' xs

maximum'' [] = error "boş listenin en büyüğü"
maximum'' [x] = x
maximum'' (x:xs) = max x (maximum' xs)
```

maximum' Fonksiyonunun Çalışması



Şekil: `maximum'` fonksiyonunu çalışması [1]

Bazı Özyineleme Örnekleri

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

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

reverse' [] = []
reverse' (x:xs) = reverse' xs ++ [x]

repeat' x = x:repeat' x

zip' _ [] = []
zip' [] _ = []
zip' (x:xs) (y:ys) = (x,y):zip' xs ys

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

uzunluk Fonksiyonu

- uzunluk fonksiyonu listedeki eleman sayısını verir
- Başka yollarla da yapılabilir fakat özyineleme için güzel bir başlangıçtır
- Kullanımı:

```
Prelude> uzunluk [1, 10, 21, 4]
4
```

`ciftFaktoryel` Fonksiyonu

- Faktoryel 1 ile n arasındaki sayıların çarpımıdır
- Çift faktoryel ise
  - Eğer n tek ise 1 ile n arasındaki tek sayıların çarpımını(7 için $1 \cdot 3 \cdot 5 \cdot 7 = 105$)
  - n çift ise 2 ile n arasındaki çift sayıların çarpımını verir(6 için $2 \cdot 4 \cdot 6 = 48$)

`palindrom` Fonksiyonu

- Palindrom kelimeler tersten ve düzden okunuşu aynı olan kelimelerdir
- Palindrom özelliği listeler için de düşünülebilir(`[1,4,5,4,1]` palindromdur)
- Parametre olarak gelen listenin palindrom olup olmadığını döndüren fonksiyonu yazın

indistekiEleman Fonksiyonu

- Parametre olarak verilen liste içindeki, yine parametre olarak verilen indisteki elemanı veren fonksiyon

Kullanımı:

```
Prelude> indistekiEleman ([1,7,5,4,6], 2)
5
```

compress Fonksiyonu

compress fonksiyonu liste içinde ardışık olarak tekrar eden elemanları bir adete dönüştürür. Kullanımı:

```
Prelude> compress "aaabbacccccccdeed"
"abacded"
Prelude> compress [1,1,1,2,2,3,1,3,3]
[1,2,3,1,3]
```

Kaynaklar I

Learn you a haskell for great good! a beginner's guide - recursion.
`http://learnyouahaskell.com/recursion`.

Learn you a haskell for great good! a beginner's guide - syntax in functions.
`http://learnyouahaskell.com/syntax-in-functions#let-it-be`.