Haskell 3

Yüksek Dereceli Fonksiyonlar

Şevket Umut ÇAKIR

PAÜ

7 Nisan 2020

Anahat

Körileme

```
ghci> max 4 5
5
ghci> (max 4) 5
5
```

Körileme

```
multThree :: (Num a) => a -> a -> a -> a
multThree x y z = x * y * z
```

Körileme

```
ghci> let multTwoWithNine = multThree 9
ghci> multTwoWithNine 2 3
54
ghci> let multWithEighteen = multTwoWithNine 2
ghci> multWithEighteen 10
180
```

Körileme

```haskell
compareWithHundred :: (Num a, Ord a) => a -> Ordering
compareWithHundred x = compare 100 x

compareWithHundred' :: (Num a, Ord a) => a -> Ordering
compareWithHundred' = compare 100
```

Yüksek Dereceli Fonksiyonlar

```
applyTwice :: (a -> a) -> a -> a
applyTwice f x = f (f x)
```

Yüksek Dereceli Fonksiyonlar

```
ghci> applyTwice (+3) 10
16
ghci> applyTwice (++ " HAHA") "HEY"
"HEY HAHA HAHA"
ghci> applyTwice ("HAHA " ++) "HEY"
"HAHA HAHA HEY"
ghci> applyTwice (multThree 2 2) 9
144
ghci> applyTwice (3:) [1]
[3,3,1]
```

Yüksek Dereceli Fonksiyonlar

```
zipWith' :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith' _ [] _ = []
zipWith' _ _ [] = []
zipWith' f (x:xs) (y:ys) = f x y : zipWith' f xs ys
```

Yüksek Dereceli Fonksiyonlar

```
ghci> zipWith' (+) [4,2,5,6] [2,6,2,3]
[6,8,7,9]
ghci> zipWith' max [6,3,2,1] [7,3,1,5]
[7,3,2,5]
ghci> zipWith' (++) ["foo ", "bar ", "baz "] ["fighters",
↪  "hoppers", "aldrin"]
["foo fighters","bar hoppers","baz aldrin"]
ghci> zipWith' (*) (replicate 5 2) [1..]
[2,4,6,8,10]
ghci> zipWith' (zipWith' (*)) [[1,2,3],[3,5,6],[2,3,4]]
↪  [[3,2,2],[3,4,5],[5,4,3]]
[[3,4,6],[9,20,30],[10,12,12]]
```

map ve filter Fonksiyonları

```
map :: (a -> b) -> [a] -> [b]
map _ [] = []
map f (x:xs) = f x : map f xs
```

map ve filter Fonksiyonları

```
ghci> map (+3) [1,5,3,1,6]
[4,8,6,4,9]
ghci> map (++ "!") ["BIFF", "BANG", "POW"]
["BIFF!","BANG!","POW!"]
ghci> map (replicate 3) [3..6]
[[3,3,3],[4,4,4],[5,5,5],[6,6,6]]
ghci> map (map (^2)) [[1,2],[3,4,5,6],[7,8]]
[[1,4],[9,16,25,36],[49,64]]
ghci> map fst [(1,2),(3,5),(6,3),(2,6),(2,5)]
[1,3,6,2,2]
```

map ve filter Fonksiyonları

```haskell
filter :: (a -> Bool) -> [a] -> [a]
filter _ [] = []
filter p (x:xs)
    | p x       = x : filter p xs
    | otherwise = filter p xs
```

map ve filter Fonksiyonları

```
ghci> filter (>3) [1,5,3,2,1,6,4,3,2,1]
[5,6,4]
ghci> filter (==3) [1,2,3,4,5]
[3]
ghci> filter even [1..10]
[2,4,6,8,10]
ghci> let notNull x = not (null x) in filter notNull
 ↪  [[1,2,3],[],[3,4,5],[2,2],[],[],[]]
[[1,2,3],[3,4,5],[2,2]]
ghci> filter (`elem` ['a'..'z']) "u LaUgH aT mE BeCaUsE I
 ↪  aM diFfeRent"
"uagameasadifeent"
```

map ve filter Fonksiyonları

```haskell
quicksort :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:xs) =
    let smallerSorted = quicksort (filter (<=x) xs)
        biggerSorted = quicksort (filter (>x) xs)
    in  smallerSorted ++ [x] ++ biggerSorted
```

map ve filter Fonksiyonları

```haskell
chain :: (Integral a) => a -> [a]
chain 1 = [1]
chain n
    | even n =  n:chain (n `div` 2)
    | odd n  =  n:chain (n*3 + 1)
```

map ve filter Fonksiyonları

```
ghci> chain 10
[10,5,16,8,4,2,1]
ghci> chain 1
[1]
ghci> chain 30
[30,15,46,23,70,35,106,53,160,80,40,20,10,5,16,8,4,2,1]
```

map ve filter Fonksiyonları

```haskell
numLongChains :: Int
numLongChains = length (filter isLong (map chain [1..100]))
    where isLong xs = length xs > 15
```

Lambda Fonksiyonlar

```
numLongChains :: Int
numLongChains = length (filter (\xs -> length xs > 15) (map
↪   chain [1..100]))
```

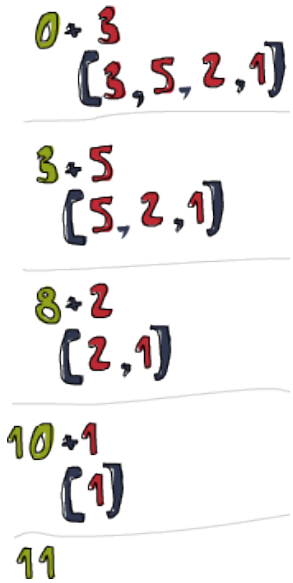
Şekil: Gordon Freeman[1]

Lambda Fonksiyonlar

```
ghci> zipWith (\a b -> (a * 30 + 3) / b) [5,4,3,2,1]
 ↪  [1,2,3,4,5]
[153.0,61.5,31.0,15.75,6.6]
ghci> map (\(a,b) -> a + b) [(1,2),(3,5),(6,3),(2,6),(2,5)]
[3,8,9,8,7]
```

fold Fonksiyonları

```haskell
sum' :: (Num a) => [a] -> a
sum' xs = foldl (\acc x -> acc + x) 0 xs
--Alternatif
sum' :: (Num a) => [a] -> a
sum' = foldl (+) 0
```



Şekil: foldl Örneği[1]

fold Fonksiyonları

```
elem' :: (Eq a) => a -> [a] -> Bool
elem' y ys = foldl (\acc x -> if x == y then True else acc)
↪  False ys

map' :: (a -> b) -> [a] -> [b]
map' f xs = foldr (\x acc -> f x : acc) [] xs
```

fold Fonksiyonları

```haskell
maximum' :: (Ord a) => [a] -> a
maximum' = foldr1 (\x acc -> if x > acc then x else acc)

reverse' :: [a] -> [a]
reverse' = foldl (\acc x -> x : acc) []

product' :: (Num a) => [a] -> a
product' = foldr1 (*)

filter' :: (a -> Bool) -> [a] -> [a]
filter' p = foldr (\x acc -> if p x then x : acc else acc) []

head' :: [a] -> a
head' = foldr1 (\x _ -> x)

last' :: [a] -> a
last' = foldl1 (\_ x -> x)
```

Kaynaklar I


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