

Функции

ИТ Кариера



Учителски екип
Обучение за ИТ кариера

<https://it-kariera.mon.bg/e-learning>

Съдържание

- Абстракции чрез функции
- Изчисления върху списъци
- Анонимни функции



Абстракции чрез функции

- Haskell предлага възможност за абстракция чрез функция
 - Ако функция `a` приема като параметри `b`, `c` и друга функция `func` и връща резултат извиканата функция `func` с параметри `b` и `c`, то резултатът всеки път ще е различен
 - Резултатът зависи от подадената функция `func`, като единственото условие е тя да приема същия брой параметри, които и се подават в тялото на `a`

Абстракции чрез функции

```
abstThroughFunction a b func = func a b  
firstFunc a b = (a * b)  
secondFunc a b = (a + b)  
thirdFunc a b = (a - b)
```

```
abstThroughFunction 10 10 firstFunc --100  
abstThroughFunction 10 10 secondFunc -- 20  
abstThroughFunction 10 10 thirdFunc  -- 0
```

Абстракции чрез функции

```
abstThroughFunction a b func = func a b
```

```
firstFunc a b = (a * b)
```

```
secondFunc a b = (a + b)
```

```
thirdFunc a b = (a - b)
```

В зависимост от функцията
резултатът е различен при едни и
същи параметри (a - 10, b - 10)

```
abstThroughFunction 10 10 firstFunc --100
```

```
abstThroughFunction 10 10 secondFunc -- 20
```

```
abstThroughFunction 10 10 thirdFunc -- 0
```


Изчисления върху списъци

- Haskell предлага много възможности за изчисления върху списъци
- Функцията `map` приема като параметри функция и списък и връща като резултат нов списък, като върху един елемент от първоначалният списък е извикана подадената функция

```
absoluteList list = map abs list  
absoluteList [1,2,-3,-4] -- [1,2,3,4]
```

```
plus1List list = map (1 + ) list  
plus1List [1,2,3,4,5] -- [2,3,4,5,6]
```

Изчисления върху списъци

- Haskell предлага много възможности за изчисления върху списъци
- Функцията `map` приема като параметри функция и списък и връща като резултат нов списък, като върху всеки елемент от първоначалният списък е извикана подадената функция

```
absoluteList list = map abs list  
absoluteList [1,2,-3,-4] == [1,2,3,4]
```

Функцията `abs` е извикана за всеки елемент от списъка - резултатът е списък с елементи само положителни числа (модули)

```
plus1List list = map (+1) list  
plus1List [1,2,3,4,5] == [2,3,4,5,6]
```

Изчисления върху списъци

- Haskell предлага много възможности за изчисления върху списъци
- Функцията `map` приема като параметри функция и списък и връща като резултат нов списък, като върху всеки елемент от първоначалният списък е извикана подадената функция

```
absoluteList list = map abs list  
absoluteList [1,2,-3,-4] -- [1,2,3,4]
```

Можем да се възползваме от факта, че операторът `+` (както и всички оператори в Haskell) е функция. Използвано е също и така нареченото частично снабдяване на параметри за функция

```
plus1List list = map (1 + ) list  
plus1List [1,2,3,4,5] -- [2,3,4,5,6]
```


Изчисления върху списъци

- Функцията `filter` тества всеки елемент от списък и връща само тези, които минават теста (функция, която връща тип boolean)

```
isEven x = x `mod` 2 == 0  
removeOdd = filter isEven
```

Изчисления върху списъци

- Функцията `filter` тества всеки елемент от списък и връща само тези, които минават теста (функция, която връща тип `boolean`)

```
isEven x = x `mod` 2 == 0  
removeOdd = filter isEven
```

Само първият аргумент на функцията `filter` е снабден - това прави функцията `removeOdd` такава, която също приема 1 аргумент - неснабденият досега

Изчисления върху списъци

- Функцията `filter` тества всеки елемент от списък и връща само тези, които минават теста (функция, която връща тип boolean)

```
isEven x = x `mod` 2 == 0  
removeOdd = filter isEven
```

- Резултат:

```
removeOdd [1,2,3,4,5,6,7,8] -- [2,4,6,8]
```

Изчисления върху списъци

- Сгъване на списък - комбиниране на всички стойности от списъка в една. Има две вградени функции, които правят това
 - Функцията ``foldl`` - действията се извършват от ляво надясно
 - Функцията ``foldr`` - действията се извършват от дясно наляво
- И двете функции приемат три параметъра
 - Акумулатор - функцията, която ще се извиква между елементите на списъка
 - Начална стойност, от която да започне изчислението
 - Самият списък
- И при двете функции изчисленията започват от стойността на акумулатора
- ``foldl`` е по-бърза функция
- ``foldr`` намира приложението си при работа с безкрайни списъци

Изчисления върху списъци

- Примери:

```
subtractList list = foldl (-) 0 list  
subtractList' list = foldr (-) 0 list
```

```
subtractList [1,2,3,4,5] -- -15  
subtractList' [1,2,3,4,5] -- 3
```


Изчисления върху списъци

- Функцията `zip` приема като аргументи два списъка и връща като резултат списък от двойки, където първият елемент е от първият списък, а другият от вторият списък

```
zip [1,3,5] [2,4,6] -- [(1,2),(3,4),(5,6)]
```

- Списъкът - резултат свършва, където свършва кой да е от подадените списъци

```
zip [1,2] [3,4,5,6] -- [(1,3),(2,4)]  
zip [] [1] -- []
```

Изчисления върху списъци

- Функцията `zipWith` освен два списъка приема и функция, която да използва при комбинирането на елементи от двата списъка

```
zipWith (+) [1,2,3,4,5] [9,8,7,6,5]  
-- [10,10,10,10,10]
```

- Вж. функциите `zipWith3`, `zipWith4` и т.н.

Задача:

- Дефинирайте функция, която приема списък и връща най-големия елемент от него
- Използвайте някоя от научените функции за изчисления върху списък

Решение:

```
maxFromList list = foldl max (head list) list  
maxFromList [-1, 5, 10] -- 10
```

Анонимни функции

- Следният синтаксис често обърква и прави кода нечетим:

```
plus3 x y z = x + y + z
```

- В такива случаи много удобни за използване са анонимните функции

```
(\x y z -> x + y + z)
```


Анонимни функции

- Следният синтаксис често обърква и прави кода нечетим

```
plus3 x y z = x + y + z
```

- В такива случаи много удобни за използване са анонимните функции

```
(\x y z -> x + y + z)
```

```
(\x y z -> x + y + z) 10 20 30 -- 60
```

Анонимни функции

- Следният синтаксис често обърква и прави кода нечетим

```
plus3 x y z = x + y + z
```

- В такива случаи много удобни за използване са анонимните функции

```
(\x y z -> x + y + z)
```

```
(\x y z -> x + y + z)
```

В Haskell анонимните функции се дефинират, като се заграждат в скоби ``()` и започват със символа ``\`` последван от параметрите, които функцията приема

Анонимни функции

- Следният синтаксис често обърква и прави кода нечетим

```
plus3 x y z = x + y + z
```

- В такива случаи много удобни за използване са анонимните функции

```
(\x y z -> x + y + z)
```

```
(\x y z -> x + y + z)
```

От дясната страна на стрелката ``->`` е резултатът, който функцията връща след изпълнението си

Анонимни функции

- Кога да използваме анонимна функция?

```
(\ x -> x ++ [(head (tail x))] ++ [head x])
```

```
(\ x -> 2 * x)
```

Анонимни функции

- Кога да използваме анонимна функция?

```
(\ x -> x ++ [(head (tail x))] ++ [head x])
```

Колкото по-дълга и сложна е една функция, толкова по-добра идея е да се напише като отделна наименована функция

```
(\ x -> 2 * x)
```


Анонимни функции

- Кога да използваме анонимна функция?

```
(\ x -> x ++ [(head (tail x))] ++ [head x])
```

Обратно - ако функцията е проста и лесно се вижда какво прави е добра идея да се напише като анонимна функция

```
(\ x -> 2 * x)
```

Анонимни функции

- Кога да използваме анонимна функция?

```
(\ x -> x ++ [(head (tail x))] ++ [head x])
```

Анонимните функции не могат да се преизползват, но ако функцията се използва само веднъж няма причина тя да не е анонимна

```
(\ x -> 2 * x)
```

Анонимни функции

- В Haskell е възможно и да се дават имена на анонимни функции, ако по някаква причина това е нужно

```
plus3' = (\ x y z -> x + y + z)
```

- Анонимните функции имат огромно приложение при използването на вградените в Haskell функции `map` и `foldl/foldr` при работа със списъци

```
addOneList list = map (\x -> x + 1) list  
addOneList [1,1,1] -- [2,2,2]
```

Задача:

- Използвайте вградената в Haskell функция `zipWith`, като за първи параметър (функция) използвате ваша анонимна функция, която връща сбора на два елемента

Решение:

- Результат:

```
zipWith (\ x y -> x + y ) [10,12] [3,4]
```

```
[13,16]
```


Обобщение

- Абстракции чрез функции
- Изчисления върху списъци
- Анонимни функции



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"



Министерство
на образованието
и науката



Национална
програма
„Обучение за
ИТ кариера“

- Курсът се разпространява под свободен лиценз **CC-BY-NC-SA**

