

# Основни понятия в Haskell

Трифон Трифонов

Функционално програмиране, 2019/20 г.

27 ноември 2019 г.

Тази презентация е достъпна под лиценза Creative Commons Признание-Некомерсиално-Споделяне на споделеното 4.0 Международен 

# Какво е Haskell?



Haskell Brooks Curry  
(1900–1982)

"Photo of Haskell B. Curry" от Gleb.svechnikov (<https://commons.wikimedia.org/wiki/File:HaskellBCurry.jpg>). CC BY-SA 4.0

# Какво е Haskell?

```

fact 0 = 1
fact n = n * fact (n-1)

quickSort []      = []
quickSort (x:xs) = quickSort less ++ [x] ++ quickSort more
  where less = filter (<=x) xs
        more = filter (>x) xs

студенти = [("Иван", 40000, 3.5), ("Мария", 60001, 5.5),
            ("Петър", 40002, 5.0), ("Галя", 40003, 4.75)]
избрани = foldr1 (++)
          [ ' ':име | (име,фн,оценка) <- студенти,
                    оценка > 4.5, фн < 60000 ]

```

# Какво е Haskell?

- Чист функционален език (без странични ефекти)
- Статично типизиран с автоматичен извод на типовете
- Използва нестриктно (лениво) оценяване
- Стандартизиран (Haskell 2010 Language Report)

## Помощни материали

- 1 S. Thompson. Haskell: The Craft of Functional Programming (2nd ed.). Addison-Wesley, 1999.
- 2 P. Hudak, Peterson J., Fasel J. A Gentle Introduction to Haskell 98, 1999 (Internet, 2008).
- 3 Haskell Wiki: <https://wiki.haskell.org/Haskell>
- 4 Haskell Platform: <https://www.haskell.org/platform/>

# Синтактични елементи

- Идентификатори: `fact`, `_muvar`, студенти
  - имена на обекти, започват с малка буква или `_`
- Запазени идентификатори: `case`, `if`, `let`, `where`, ...
- Конструктори: `Integer`, `Maybe`, `Just`, ...
  - имена на конструкции, започват с главна буква
- Числа: `10`, `-5.12`, `3.2e+2`, `1.2E-2`, `0x2f`, `0o35`
- Операции: `+`, `*`, `&%`, `<=>`, ♠
  - поредица от символи (без букви и цифри)
  - всички операции с изключение на унарния - са инфиксни
- Запазени операции: `..` `:` `::` `=` `\|` `<-` `->` `@` `~` `=>`
- Специални символи: `( )` `,` `;` `[ ]` `'` `{ }`
- Знаци: `'a'`, `'\n'`, `'+'`
- Низове: `"Hello, world!"`, "произволен низ"

# Декларации и дефиниции

- `<име> :: <тип>` (типова декларация)
- декларира се, че `<име>` ще се свързва със стойности от `<тип>`
- **типовите декларации са незадължителни: в повечето случаи Haskell може сам да се ориентира за правилния тип**
  - `x :: Int`
  - `y :: Double`
  - `z :: String`
- `<име> = <израз>` (дефиниция)
- `<име>` се свързва с `<израз>`
  - `x = 2`
  - `y = fromIntegral x^2 + 7.5`
  - `z = "Hello"`
  - ~~`z = x + y`~~

# Типове

Типовете в Haskell обикновено се задават с конструктори.

- `Bool` — булев тип с константи `True` и `False`
- `Char` — Unicode знаци
- Целочислени
  - `Int` — цели числа с фиксирана големина  $[-2^{63}; 2^{63} - 1]$
  - `Integer` — цели числа с произволна големина
- С плаваща запетая
  - `Float` — дробни числа с единична точност
  - `Double` — дробни числа с двойна точност
- Съставни
  - `[a]` — тип списък с **произволна** дължина и елементи от **фиксиран** тип `a`
  - `String = [Char]` — низ (списък от знаци)
  - `(a,b,c)` — тип кортеж (наредена  $n$ -торка) с **фиксирана** дължина и **произволни** типове на компонентите



# Стандартен модул Prelude

- програмите в Haskell се разделят на модули
- `module <име> where`
- дефинира модул с <име>
- `import <модул> [( <име> { , <име> } )]`
- внася дефинициите <име> от <модул>
- ако <име> не е указано, внася всички дефиниции
- модулът `Prelude` съдържа набор от често използвани стандартни функции
- всички дефиниции от `Prelude` се внасят автоматично във всяка програма

# Стандартни числови функции

Аритметични операции

`+`, `-`, `*`, `/`, `^`, `^^`

Други числови функции

`div`, `mod`, `max`, `min`, `gcd`, `lcm`

Функции за преобразуване

`fromIntegral`, `fromInteger`, `toInteger`, `realToFrac`, `fromRational`,  
`toRational`, `round`, `ceiling`, `floor`

Функции над дробни числа

`exp`, `log`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `sqrt`, `**`

# Стандартни предикати

Числови предикати

<, >, ==, /=, <=, >=, `odd`, `even`

Булеви операции

`&&`, `||`, `not`

# Функции в Haskell

- $t1 \rightarrow t2$  — тип на функция, която получава параметър от тип  $t1$  и връща резултат от тип  $t2$
- `<име> <параметър> = <тяло>`
- дефиниция на функция с `<име>`, един `<параметър>` и `<тяло>`
- `<функция> <израз>`
- прилагане на `<функция>` над `<израз>`
  - `square :: Int -> Int`
  - `square x = x * x`
  - `square x`  $\rightarrow$  4
  - `square 2.7`  $\rightarrow$  Грешка!
- Прилагането е с по-висок приоритет от другите операции!
- `square 2 + 3`  $\rightarrow$  7
- `square (2 + 3)`  $\rightarrow$  25

## Функции на повече параметри

- Как можем да изразим двуаргументна функция  $f(x, y)$ , ако разполагаме само с едноаргументни функции?
- Разглеждаме функция  $F$  с един аргумент  $x, \dots$
- ...която връща като резултат едноаргументната  $f_{x, \dots}$
- ...така че  $f_x(y) = f(x, y)$ .
- Така имаме  $f(x, y) = F(x)(y)$ .

### Основна идея

Можем да разглеждаме функция с  $n + 1$  аргумента, като функция с един аргумент, която връща функция с  $n$  аргумента.

Това представяне на функциите с повече аргументи се нарича “къринг” (“currying”).

## Currying в Haskell

- $t1 \rightarrow (t2 \rightarrow t3)$ 
  - функция с параметър от тип  $t1$ , която връща функция, която приема параметър от тип  $t2$  и връща резултат от тип  $t3$ ; или
  - функция на два параметъра от типове  $t1$  и  $t2$ , която връща резултат от тип  $t3$
- В общия случай:  $\langle \text{функция} \rangle :: t1 \rightarrow (t2 \rightarrow \dots (tn \rightarrow t) \dots)$
- $\langle \text{функция} \rangle$  ще очаква  $n$  параметъра от типове  $t1, t2, \dots, tn$  и ще връща резултат от тип  $t$
- $\langle \text{функция} \rangle \langle \text{параметър}_1 \rangle \dots \langle \text{параметър}_n \rangle = \langle \text{тяло} \rangle$
- дефинира  $\langle \text{функция} \rangle$  с  $n$  параметъра и  $\langle \text{тяло} \rangle$ 
  - `hypotenuse :: Double -> Double -> Double`
  - `hypotenuse a b = sqrt (a**2 + b**2)`
  - `hypotenuse 3 4 → 5`

## Частично прилагане на функции

Кърингът позволява удобно прилагане на функция към само част от параметрите.

- `div50 :: Int -> Int`
- `div50 x = div 50 x`
- `div50 4 → 12`

## Функции от по-висок ред

**Внимание:**  $t1 \rightarrow (t2 \rightarrow t3) \neq (t1 \rightarrow t2) \rightarrow t3!$

- операцията  $\rightarrow$  е дясноасоциативна
- $t1 \rightarrow (t2 \rightarrow t3) \equiv t1 \rightarrow t2 \rightarrow t3$
- $(t1 \rightarrow t2) \rightarrow t3$  — функция, която връща резултат от тип  $t3$ , а приема като единствен параметър функция, която приема един параметър от тип  $t1$  и връща резултат от тип  $t2$
- **функция от втори ред**
  - `twice f x = f (f x)`
  - `twice :: (Int -> Int) -> Int -> Int`
  - `twice square 3`  $\longrightarrow$  81
  - `twice (mod 13) 5`  $\longrightarrow$  1
  - `diag f x = f x x`
  - `diag :: (Int -> Int -> Int) -> Int -> Int`
  - `diag div 5`  $\longrightarrow$  1
  - `diag hypotenuse 1`  $\longrightarrow$  1.4142135623730951



# Функции и операции

- Функциите в Haskell са винаги с **префиксен** запис
- Операциите в Haskell са винаги **бинарни** с **инфиксен** запис.
  - **Изключение:** унарен минус: `-a`
  - `square -x` → **Грешка!**
  - `square (-x)` → 4
- Преобразуване на двуаргументни функции към бинарни операции:  
'<функция>'
  - `13 'div' 5` → 3
  - `2 'square'` → **Грешка!**

# Операции и функции

- Преобразуване на операции към двуаргументни функции:  
(`<операция>`)
  - `(+)`  $2\ 3 \longrightarrow 5$
  - `plus1 = (+)`  $1$
  - `square = diag` `(*)`
- Преобразуване на операции към едноаргументни функции  
(отсичане на операции)
  - `(<израз> <операция>)` — ляво отсичане
  - `(<операция> <израз>)` — дясно отсичане
  - `(2^)`  $3 \longrightarrow 8$
  - `(^2)`  $3 \longrightarrow 9$
  - `square = (^2)`
  - `(-5)`  $8 \longrightarrow$  **Грешка!**
  - `twice (*2)`  $5 \longrightarrow 20$
  - `positive = (>0)`
  - `lastDigit = ('mod' 10)`

# if ... then ... else

- `if <условие> then <израз1> else <израз2>`
  - Ако <условие> е `True`, връща <израз<sub>1</sub>>
  - Ако <условие> е `False`, връща <израз<sub>2</sub>>
- `abs x = if x < 0 then -x else x`
- `fact n = if n == 0 then 1 else n * fact (n - 1)`
- `if x > 5 then x + 2 else "Error"` → Грешка!
- <израз<sub>1</sub>> и <израз<sub>2</sub>> трябва да са от един и същи тип!
- <условие> трябва да е от тип `Bool`!