# Основни понятия в програмирането на езика Haskell

## Общи сведения за езика Haskell

Haskell е език за строго функционално програмиране. Създаден е в края на 80-те години на 20-ти век. Носи името на Haskell B. Curry – един от пионерите на λ-смятането (математическа теория на функциите, дала тласък в развитието на множество езици за функционално програмиране).

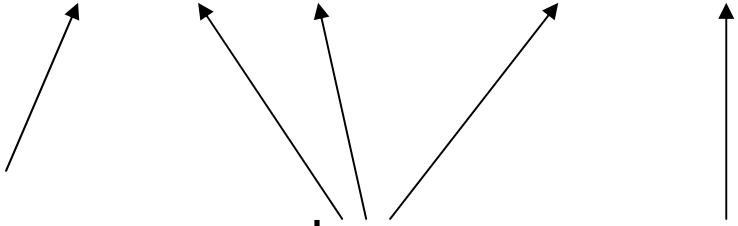
Най-популярната среда за програмиране на Haskell е **Hugs** (1998). Тя предоставя много добри средства за обучение и се разпространява безплатно за множество платформи.

Haskell home page:
http: //www.haskell.org/

## Дефиниране на функции

Общ вид на дефиниция на функция:

```
name x1 x2 ... xk = e
```



име на функцията

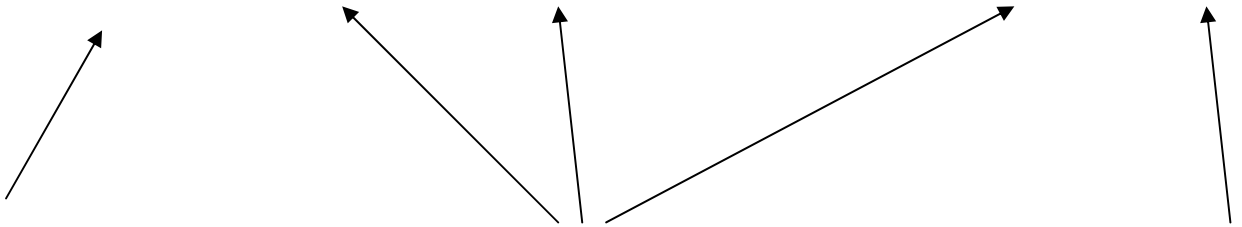
формални параметри

тяло на функцията (израз, дефиниращ резултата, който връща функцията)

Дефиницията на функция трябва да бъде предшествана от декларация на нейния тип (на типовете на аргументите и типа на резултата, който връща функцията).

Общ вид на декларация на типа на функция:

```
name :: t1 -> t2 -> ... -> tk -> t
```



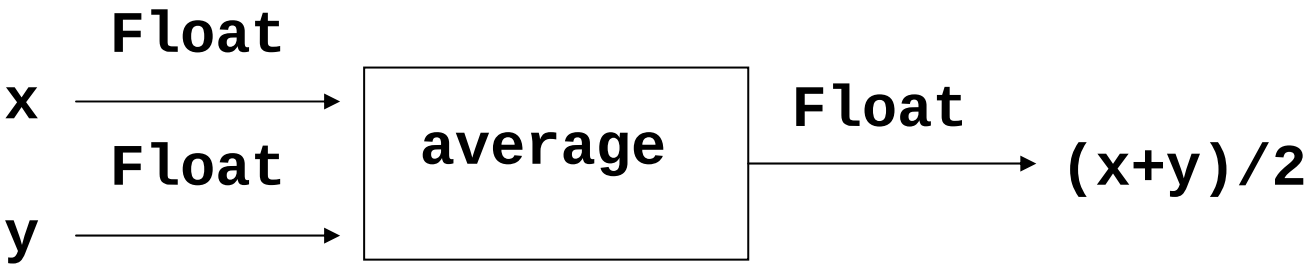
Примери

```
square :: Int -> Int
square n = n*n

> square 3
9
> square 5
25
```

```
average :: Float -> Float -> Float
average x y = (x+y)/2

> average 3.4 5.6
4.5
> average 3 4
3.5
```



## Общ вид на програмата на Haskell

Програмите на Haskell обикновено се наричат наричат **скрипт**ове (**script**s). Освен програмния код (поредица от дефиниции на функции) един скрипт може да съдържа и коментари.

Има два различни стила на писане на скриптове, които съответстват на две различни философии на програмиране.

Традиционно всичко в един програмен файл (файл с изходния код на програма на Haskell) се интерпретира като програмен текст (код), освен ако за нещо е отбелязано специално, че представлява коментар. Скриптовете, написани в такъв (traditional) стил, се съхраняват във файлове с разширение “.hs”.

Традиционно коментари се означават по два начина. Символът “--“ означава начало на коментар, който продължава от съответната позиция до края на текущия ред. Коментари, които съзържат произволен брой знакове и евентуално заемат повече от един ред, могат да бъдат заключени между символите “{-“ и “-}”.

Алтернативният (literate) подход предполага, че всичко във файла е коментар освен частите от текста, специално означени като програмен код. В Пример 2 програмният текст е само в редовете, започващи с “>” и отделени от останалия текст с празни редове.

Този вид скриптове се съхраняват във файлове с разширение “.lhs”.

Пример 1. A traditional script

```
{- ##########################################################
     MyFirstScript.hs
########################################################## -}

-- The value size is an integer (Int), defined to be
-- the sum of 12 and 13.

size :: Int
size = 12+13

-- The function to square an integer.

square :: Int -> Int
square n = n*n
```

```
-- The function to double an integer.

double :: Int -> Int
double n = 2*n

-- An example using double, square and size.

example :: Int
example = double (size – square (2+2))
```

Пример 2. A literate script

```
{- ##########################################################
     MyFirstLiterate.lhs
########################################################## -}

The value size is an integer (Int), defined to be
the sum of 12 and 13.

>   size :: Int
>   size = 12+13

The function to square an integer.

>   square :: Int -> Int
>   square n = n*n
```

```
The function to double an integer.

>   double :: Int -> Int
>   double n = 2*n

An example using double, square and size.

>   example :: Int
>   example = double (size – square (2+2))
```

# Библиотеки на Haskell

Haskell поддържа множество вградени типове данни: цели и реални числа, булеви стойности, низове, списъци и др., както и предлага вградени функции за работа с данни от тези типове.

Дефинициите на основните вградени функции в езика се съдържат във файл (**the standard prelude**) с името Prelude.hs. По подразбиране при стартиране на Hugs (или друга среда за програмиране на Haskell) най-напред се зарежда the standard prelude, след което потребителят може да започне своята работа.

Напоследък, с цел намаляване на обема на the standard prelude, дефинициите на част от вградените функции се преместват от the standard prelude в множество **стандартни библиотеки**, които могат да бъдат включени от потребителя в средата на Haskell при необходимост.

# Модули

Възможно е текстът на една програма на Haskell да бъде разделен на множество компоненти, наречени **модули**.

Всеки модул има свое **име** и може да съдържа множество от дефиниции на Haskell. За да се дефинира даден модул, например Aut, е необходимо в началото на програмния текст в съответния файл да се включи ред от типа на

```
module Aut where
  .....
```

Един модул може да **импортира** дефиниции от други модули. Например модулът Bee ще може да импортира дефиниции от модула Aut чрез включване на оператор import както следва:

```
module Bee where
import Aut
    .....
```

В случая операторът import означава, че при дефинирането на функции в Bee могат да се използват всички (**видими**) дефиниции от Aut.

Механизмът на модулите поддържа споменатите по-горе библиотеки.

Механизмът на модулите позволява да се определи кои дефиниции да бъдат достъпни чрез **експортиране** от даден модул за употреба от други модули.

# Основни типове данни в Haskell

## Булеви стойности (Bool)

Булевият тип в Haskell се нарича Bool. Булеви константи са True и False, а вградените Булеви оператори, поддържани от езика, са:

| | |
|---|---|
| && | and |
| \|\| | or |
| not | not |

## Цели числа (Int и Integer)

Целите числа (числата с фиксирана точка) в Haskell са от тип Int или Integer. За представяне на целите числа от тип Int се използва фиксирано пространство (32 бита), което означава, че типът Int съдържа краен брой елементи. Константата **maxBound** има за стойност максималното число от тип Int ($2^{32}$-1 = 2147483647).

За работа с цели числа с неограничена точност може да бъде използван типът Integer.

Haskell поддържа следните вградени оператори за работа с цели числа:

| | |
|---|---|
| + | Сума на две цели числа. |
| * | Произведение на две цели числа. |
| ^ | Повдигане на степен; 2^3 е 8. |
| - | Разлика на две цели числа (при инфиксна употреба: a-b) или унарен минус (при префиксна употреба: -a). |
| div | Частно при целочислено деление, например div 14 3 е 4. Може да се запише и инфиксно: 14 `div` 3. |
| mod | Остатък при целочислено деление, например mod 14 3 или 14 `mod` 3. |
| abs | Абсолютната стойност на дадено цяло число (числото без неговия знак). |
| negate | Функция, която променя знака на дадено цяло число. |

Забележка. Чрез заграждане на името на всяка двуаргументна функция в обратни апострофи (backquotes) е възможно записът на обръщението към тази функция да стане инфиксен.

Вградени оператори за сравнения:

| | |
|---|---|
| > | greater than |
| >= | greater than or equal to |
| == | equal to (може да се използва и при аргументи от други типове) |
| /= | not equal to |
| <= | less than or equal to |
| < | less than |

## Реални числа (числа с плаваща точка, Float)

За представянето на числата с плаваща точка в Haskell се използва фиксирано пространство, което рефлектира върху точността на работата с този тип числа.

Допустим запис:

- като десетични дроби, например

0.31426
-23.12
567.345
4513.0

- scientific notation (запис с мантиса и порядък), например

231.61e7 $231.61 \times 10^{7} = 2316100000$
231.61e-2 $231.61 \times 10^{-2} = 2.3161$
-3.412e03 $-3.412 \times 10^{3} = -3412$

Някои вградени аритметични оператори:

| | |
|---|---|
| + - * | Float -> Float -> Float |
| / | Float -> Float -> Float |
| ^ | Float -> Int -> Float ($x^n$ за неотрицателно цяло n) |
| ** | Float -> Float -> Float ($x^y$) |
| == /= < | Float -> Float -> Bool |
| > <= >= | Float -> Float -> Bool |
| signum | Float -> Float (връща резултат 1.0, 0.0 или –1.0) |
| sqrt | Float -> Float |

## Знакове (characters, Char)

Представляват отделни знакове, заградени в единични кавички (апострофи), например ‘d’ или ‘3’.

Връзка между знаковете и техните ASCII кодове:

```
ord :: Char -> Int
chr :: Int -> Char
```

Конвертирането на малки букви към главни изисква към съответния код да бъде прибавено определено отместване:

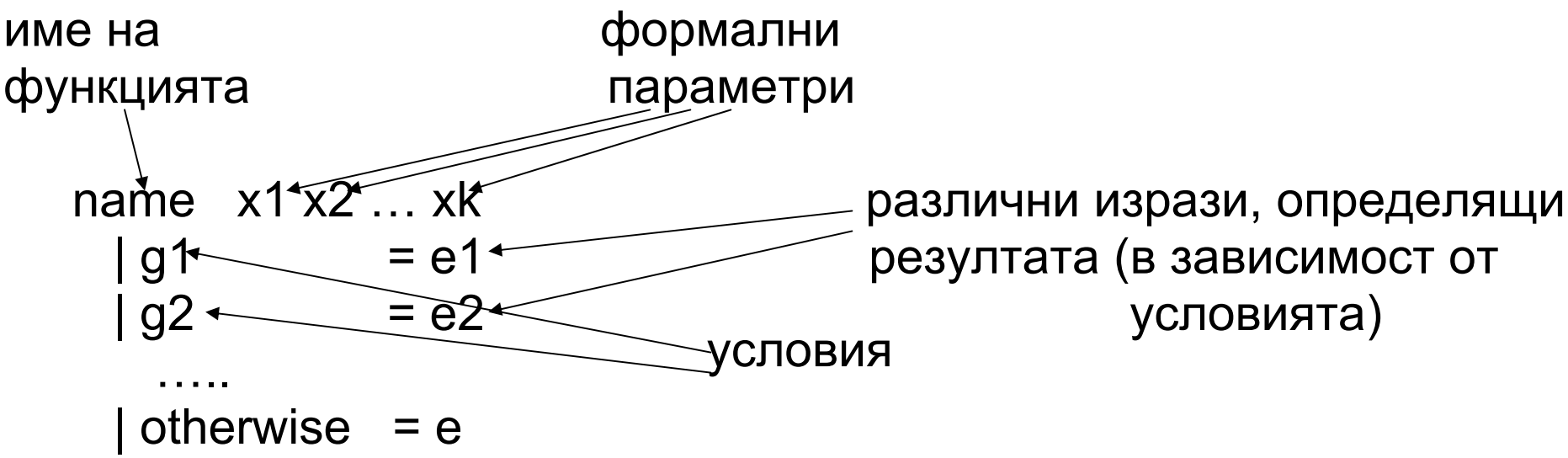
```
offset :: Int
offset = ord ‘A’ – ord ‘a’

toUpper :: Char -> Char
toUpper ch = chr (ord ch + offset)
```

## Програмиране на условия (guards)

Условието (guard) е Булев израз. Условия се използват, когато трябва да се опишат различни случаи в дефиницията на функция. Съществува аналогия между условията (guards) в Haskell и специалната форма cond в Scheme.

Общ вид на дефиниция на функция с условия:



Забележка. Клаузата otherwise не е задължителна.

Примери

```
max :: Int -> Int -> Int
max x y
    | x >= y     = x
    | otherwise  = y

maxThree :: Int -> Int -> Int -> Int
maxThree x y z
    | x >= y && x >= z   = x
    | y >= z             = y
    | otherwise          = z
```

```
fact :: Int -> Int
fact n
   | n == 0   = 1
   | n > 0    = fact (n-1) * n
```

## Условни изрази

Аналогично на Scheme и в езика Haskell е възможно да се описват условни изрази в термините на конструкцията
if … then … else.

Общ вид на условен израз в Haskell:

```
if condition then m else n
```

Пример

```
max :: Int -> Int -> Int
max x y
   = if x >= y then x else y
```