

# ЛЕНИВЫЕ ВЫЧИСЛЕНИЯ:

ПЛЮСЫ И МИНУСЫ

(RuHaskell.Meetup 2015 Summer, fromGregorian 2015 06 21)

” *Лучше ничем не заниматься,  
чем заниматься ничем.*

Тацит

**QUI EST QUI**



Haskell-программа –  
совокупность **выражений**

/англ. **expression**/

12.3 - 1.09

Это выражение

```
"/home/dsh" </> "meetup2015"
```

И это

```
head [15, 2, 6]
```

И это

9

И даже это!



Запуск Haskell-программы —  
начало **вычисления** её выражений

/англ. **evaluation**/

Вычислить выражение –  
значит **сократить** его

/англ. **reduce**/

В основе сокращения –  
применение функции  
/англ. **function application**/

```
let sum = 2 + 3
```

Можно сократить

```
let sum = (+) 2 3
```

Сокращаем...

```
let sum = 5
```

ГОТОВО

Если сократить нельзя —  
выражение в **нормальной форме**

/англ. **normal form**/

Если сократить можно —  
выражение называют **сокращаемым**  
/англ. **reducible expression** или **redex**/



# ПРИНЦИПЫ ЛЕНИ



 № 1

Выражение не сокращается,  
пока в этом нет необходимости

```
main :: IO ()  
main =  
    let stupid = 2 `div` 0  
    in print stupid
```

Вполне ожидаемая ошибка

```
main :: IO ()
main =
    let stupid = 2 `div` 0
    in putStrLn "Just exit!"
```

Никакой ошибки

 № 2

Спускаемся настолько,  
насколько это нужно

```
main :: IO ()
main =
  let some          = ( 2 `div` 0
                      , [2 * 3, 4 `div` 0]
                      )
      (_, second)  = some
      [elem1, _]   = second
  in print elem1
```

Ленивый лифт

ПЛЮСЫ



 № 1

Рациональность



```
main :: IO ()
main = do
    file <- readFile "/home/dsh/big.log"
    putStrLn $ take 100 file
```

Читаем лишь то, что нужно

 № 2

Бесконечность

```
head :: [a] -> a  
head (x:_) = x
```

Возвращаем первый элемент списка

```
main :: IO ()  
main =  
    print $ head [1, 10, 100]
```

Причём как конечного списка...

```
main :: IO ()  
main =  
    print $ head [1..]
```

... так и бесконечного

 № 3

DSL

```
selectBy :: Bool -> (a, a) -> a  
selectBy True  (f, _) = f  
selectBy False (_, s) = s
```

Замена условной конструкции

```
import System.Exit

main :: IO ()
main = do
    putStrLn "Input web prefix:"
    prefix <- getLine
    selectBy (prefix == "https")
        ( putStrLn "secure web :)"
        , exitWith $ ExitFailure 1
        )
```

И это честная замена!



# МИНУСЫ



 № 1

Неожиданное поведение

```
import Data.String.Utils

main :: IO ()
main = do
    file <- readFile path
    writeFile path $ replace "," ";" file
    where path = "/home/dsh/data"
```

Hy-ny...

```
openFile: resource busy (file is locked)
```

Лень мешает

```
import Data.String.Utils
import qualified System.IO.Strict as S

main :: IO ()
main = do
    file <- S.readFile path
    writeFile path $ replace "," ";" file
    where path = "/home/dsh/data"
```

Строгость решает

Выход:

Не игнорируйте слова  
**strictly** и **lazily**  
в документации

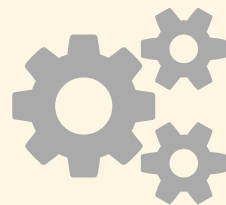
 № 2

Space Leak

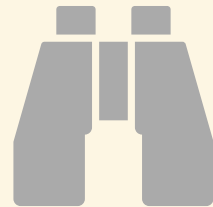
Не путать с Memory Leak!



Memory Leak –  
низкоуровневая ошибка  
управления памятью



Space Leak –  
высокоуровневая ошибка  
проектирования



```
main :: IO ()
main =
    let stupid = 2 `div` 0
    in putStrLn "Just exit!"
```

Деления не было

```
2 `div` 0
```

Невычисленное (отложенное) выражение

/англ. **unevaluated expression** или **thunk**/

Все отложенные выражения

живут в памяти

Thank you Cap!

Проблема в их количестве!



```
f :: Num b => [a] -> b -> b
f []      c = c
f (_:xs) c = f xs $ c + 1
```

Пустышка с большой проблемой

```
f [1,2,3] 0
```

Применяем...



```
f 1:[2,3] 0 + 1
```

Первый шаг...

```
f 1:2:[3] (0 + 1) + 1
```

Второй шаг...

```
f 1:2:3:[] ((0 + 1) + 1) + 1
```

Третий шаг...

```
f [] ((0 + 1) + 1) + 1
```

Последний шаг...

$$((0 + 1) + 1) + 1 = 3$$

ГОТОВО

# Ленивый вариант

```
main :: IO ()
main =
    let v = f [1..50000000] (0 :: Integer)
    in print v
```

 Время: 62 с

 Память: 6.19 ГБ


```
f' :: Num b => [a] -> b -> b
f' []      c = c
f' (_:xs) c = f' xs $! c + 1
```

Пустышка без проблем

# Строгий вариант

```
main :: IO ()
main =
    let v = f' [1..50000000] (0 :: Integer)
    in print v
```

 Время: 4 с

 Память: мизерная



Выход:

Если код может раздуться,  
разбавьте лень строгостью

```
main :: IO ()
main =
  print $ Laziness good bad
  where good = [ "Рациональность"
                , "Бесконечность"
                , "DSL"
                ]
          bad  = [ "Неожиданное поведение"
                  , "Space Leak"
                  ]
```

Благодарю за внимание!

Денис Шевченко

 [dshevchenko.biz](https://dshevchenko.biz)