Funkcionálne programovanie 4

Peter Borovanský, KAI, I-18,
borovan(a)ii.fmph.uniba.sk

- lenivé výpočty (lazy vs. eager evaluation strategy)
- nekonečné množiny/postupnosti

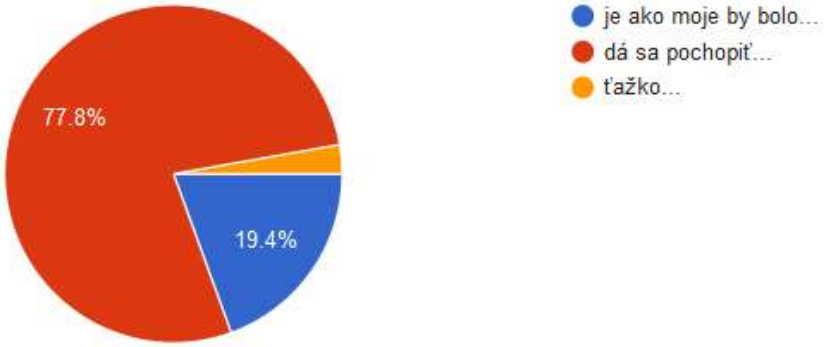
Kto chce pokračovať, je vítaný
http://dai.fmph.uniba.sk/courses/FPRO/

Anketa

(slová dĺžky k nad abecedou {A,B,C,D,E,F})

To riešenie

36 responses



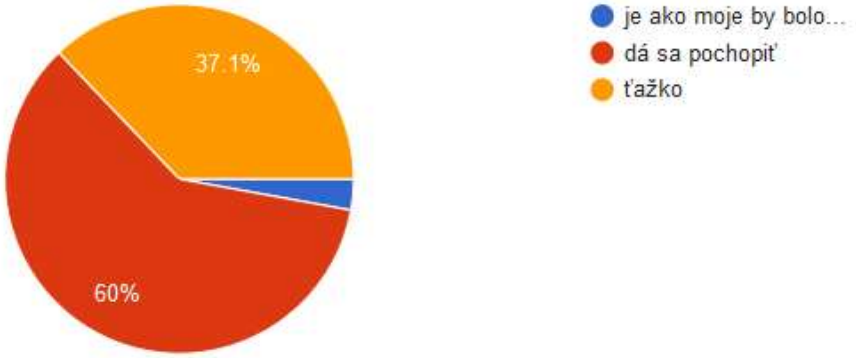

```
def words(k, current = ''):
  if len(current) == k:
    return [current]
  result = []
  for ch in 'ABCDEF':
    result += words(k, current + ch)
  return result
print(words(3))
```

slova1.hs

Anketa
(list comprehension)

Toto riešenie

35 responses




```
def words(len):
        if len == 0:
                return [""]
        else:
                return [ ch+slovo
                         for slovo in words(len-1)
                         for ch in 'ABCDEF']
print(words(3))
```

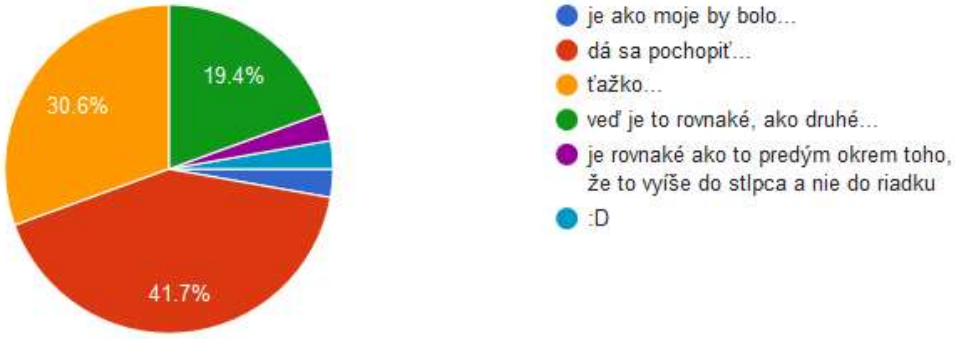
slova2.hs

Anketa
(generátor)

```
def words(len):
        if len == 0:
                return [""]
        else:
                return [ ch+slovo
                         for slovo in words(len-1)
                         for ch in 'ABCDEF']
print(words(3))
```

Toto riešenie

36 responses




```
def words(len):
        if len == 0:
                return [""]
        else:
                return (ch+slovo
                        for slovo in words(len-1)
                        for ch in 'ABCDEF')
for m in words(3):
        print(m)
```

slova3.hs

O čom boli generátory ?

```
def words(len):
        if len == 0:
                return [""]
        else:
                return [ ch+slovo
                  for slovo in words(len-1)
                  for ch in 'ABCDEF']
print(words(3))
```

# a čo toto...
print(words(8))

```
def words(len):
        if len == 0:
                return [""]
        else:
                return (ch+slovo
                 for slovo in words(len-1)
                 for ch in 'ABCDEF')
for m in words(3):
        print(m)
```

for m in words(8):
    print(m)

Variácie na variácie

- **Klauzálna definícia:**

~~slova 0 = [ ]~~

```
slova 0 = [ [] ]                -- to isté ako slova 0 = [ " " ]
slova k = [  ch:w | w <- slova (k-1), ch <- "ABCDEF" ]
```

- **Aritmetický pattern** už nie je podporovaný:

slova ~~**(k+1)**~~ = [ch:w | w <- slova k, ch <- "ABCDEF"]

- **Guards** alias bachari, či strážci:

```
slova k | k == 0      = [ [] ]
slova   | otherwise = [  ch:w | w <- slova (k-1), ch <- "ABCDEF" ]
```

- **where** patrí klauzule a nie je to výraz:

```
slova k | k == 0      = [ [] ]
slova   | otherwise = [  ch:w | w <- ws, ch <- "ABCDEF" ]
                            where ws = slova (k-1)
```

slova.hs

Slová dĺžky najviac k

```
module Slova where
import Data.List   -- pozrite si, koľko užitočných funkcií obsahuje

slova   :: Int -> [String]
slova   0   = [[]]
slova   k = [ ch:w | w <-slova (k-1), ch <- "ABCDEF"]

slova'  :: Int -> [String]
slova'  0   = [[]]
slova'  k = slova' (k-1) ++ [ ch:w | w <-slova' (k-1), ch <- "ABCDEF"]
```

length $ slova 3 = 216

length $ slova' 2 = 49 != 1+6+36 = 43
slova' 2 =
["","A","B","C","D","E","F","A","B","C","D
DA","EA","FA","AB","BB","CB","DB","EB",
,"EC","FC","AD","BD","CD","DD","ED","FD
E","FE","AF","BF","CF","DF","EF","FF"]

O(n^2). The nub function removes duplicates. The name nub means `essence'.)

length $ nub $ slova' 2 = 43

koľko je $1+6+36+...+6^k$ (počet slov dĺžky najviac k) ?

[1,7,43,259,1555,9331,55987,335923,2015539,12093235,72559411, ...]

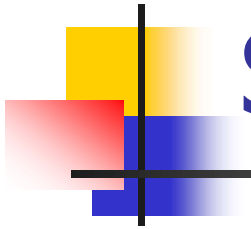
where:

```
slova'' k = ws ++ [ ch:w | w <-ws, ch <- "ABCDEF"]  where ws = slova'' (k-1)
```

let:

```
slova''' k = let ws = slova''' (k-1) in ws ++ [ ch:w | w <-ws, ch <- "ABCDEF"]
```

slova.hs

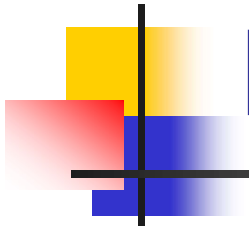


Slová dĺžky najviac k

[1,7,43,259,1555,9331,55987,335923,2015539,12093235,72559411, ...]

```
*Slova> length $ nub $ slova" 5          9331      (1.25 secs, 5,044,680 bytes)
*Slova> length $ nub $ slova" 6          55987     (39.98 secs, 33,990,296 bytes)

*Slova> length $ slovaDlzkyNajviac  5    9331      (0.01 secs, 2,933,344 bytes)
*Slova> length $ slovaDlzkyNajviac  6    55987     (0.03 secs, 17,267,680 bytes)
*Slova> length $ slovaDlzkyNajviac  7    335923    (0.13 secs, 103,265,176 bytes)
*Slova> length $ slovaDlzkyNajviac  8    2015539   (0.82 secs, 619,244,920 bytes)
*Slova> length $ slovaDlzkyNajviac  9    12093235  (4.55 secs, 3,715,116,024 bytes)
*Slova> length $ slovaDlzkyNajviac  10   72559411  (27.21 secs, 22,290,328,280 bytes)
```



Koncept generátorov

- sa v rôznych jazykoch volá a programuje rôzne
- ale v zásade ide o koncept mať časť výsledku a spôsob ako pokračovať vo výpočte

- Continuations (Lisp, C#, …)
  https://en.wikipedia.org/wiki/Continuation

- generator (Python, Rust …)

- Flow (Kotlin)

- laziness (Haskell)

Nekonečno – výhoda lenivých

lenivý (lazy) výpočet vyhodnocuje len tie výrazy:

- ktorých hodnotu naozaj treba pre ďalší výpočet,
- a navyše len raz, ak sa výrazy opakujú.

- foo 0 x = 0
  foo (n+1) x = 1+x

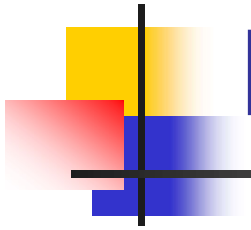
```
Main> foo 0 (5 `div` 0)
0
Main> foo 1 (5 `div` 0)
Program error: divide by zero
```

- ones = 1 : ones
- cycle :: [a] -> [a]
  ones' = cycle [1]

```
Main>ones
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
```

- goo 0 _ = 0
  goo (n+1) x = length x

```
Main> goo 0 ones
0
Main> goo 1 ones
{Interrupted!}
```



Nekonečné postupnosti

- numsFrom n = n : numsFrom (n+1)

```
Main>numsFrom 2
[2,3,4,5,6,7,8,9,10,11,12,13,
```

- squares = map (^2) (numsFrom 0)

```
Main> take 10 squares
[0,1,4,9,16,25,36,49,64,81]
```

- fibonacci :: Integer -> Integer -> [Integer]

  fibonacci a b = a : (fibonacci b (a+b))

```
Main> take 10 (fibonacci 1 1)
[1,1,2,3,5,8,13,21,34,55]

Main> (fibonacci 1 1)!!15
987
```

https://www.wolframalpha.com/input/?i=fibonacci+2000

```
take                    :: Int -> [a] -> [a]
take n _  | n <= 0      = []
take _ []               = []
take n (x:xs)           = x : take (n-1) xs
```

Ako to funguje ?

- čo sa počíta, keď zadáme fibonacci 1 1

  nič (začne sa to počítať, až keď výsledok chcete použiť, či zobraziť)

- a čo, keď head (fibonacci 1 1)

  head (x:_) = x

  fibonacci 1 1 = 1:(fibonacci 1 (1+1)) -- viac nemusím počítať

  teda,

  head (1:(fibonacci 1 (1+1))) = 1

- take 3 (fibonacci 1 1)

  take 3 1:(fibonacci 1 (1+1))=
  1:take (3-1) (fibonacci 1 (1+1)) =
  1:take 2 (1:fibonacci (1+1) (1+(1+1))) =
  1:1:take 1 (fibonacci (1+1) (1+(1+1))) =
  1:1:take 1 ((1+1):fibonacci (1+(1+1)) ((1+1)+(1+(1+1)))) =
  1:1:(1+1):take 0 (fibonacci (1+(1+1)) ((1+1)+(1+(1+1)))) =
  1:1:(1+1):[] = 1:1:2

Operácie nad ‘nekonečnom’

- mocniny2 = 1:[2^x | x <- [1..]]
  mocniny3 = 1:[3*x | x <- mocniny3]

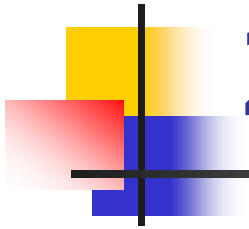
```
Main> take 7 mocniny2
[1,2,4,8,16,32,64]
Main> take 7 mocniny3
[1,3,9,27,81,243,729]
Main> take 7 mp
[1,2,3,4,8,9,16]
```

- mp = tail (merge mocniny2 mocniny3)

čo ak by tu bol append ???

- merge [] x = x
  merge x [] = x
  merge z1@(a:b) z2@(c:d) = if a < c then a:(merge b z2)
  else c:(merge z1 d)
- take 7 mp

```
take                    :: Int -> [a] -> [a]
take n _  | n <= 0      = []
take _ []               = []
take n (x:xs)           = x : take (n-1) xs
```

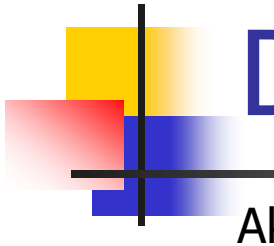


2^n, 2^n-1, 2^n+1

```
dveNaNtu                  = map (2^) [1..]
dveNaNtuMinusJedna        = map (+(-1)) dveNaNtu
dveNaNtuMinusJedna'       = 1:[2*x+1 | x <- dveNaNtuMinusJedna']
                                                  [1,3,7,15,31,63,127,255,511,1023]
dveNaNtuPlusJedna         = map (+2) dveNaNtuMinusJedna
                                                  [3,5,9,17,33,65,129,257,513,1025]
dveNaNtuMinusPlusJedna = map2 (+) dveNaNtuMinusJedna dveNaNtuPlusJedna
                                                  [4,8,16,32,64,128,256,512,1024,2048]
map2 f [] [] = []
map2 f (a:as) (b:bs) = (f a b):map2 f as bs

Main> take 20 dveNaNtuMinusPlusJedna
[4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384,32768,65536,131072,26
    2144,524288,1048576,2097152]
```

čo by urobilo: [a+b | a<-dveNaNtuMinusJedna, b<-dveNaNtuPlusJedna] ???
Skúsme: take 10 ([(a,b) | a<-dveNaNtuMinusJedna, b<-dveNaNtuPlusJedna])



Dvojice prirodzených čísel

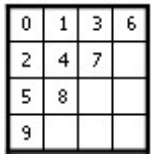
Ako dostaneme nekonečný zoznam dvojíc prirodzených cisel

dvojice = [(i,j) | i<-[0..], j<-[0..]] ☹

```
Main> take 20 dvojice
[(0,0),(0,1),(0,2),(0,3),(0,4),(0,5),(0,6),(0,7),(0,8),(0,9),(0,10),(0,11),(0,12),(0,13),
    (0,14),(0,15),(0,16),(0,17),(0,18),(0,19)]
```

diagonálne

dvojice' = [(i,s-i) | s<-[0..], i<-[0..s]]

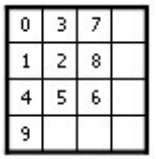



```
Main> take 20 dvojice'
[(0,0),(0,1),(1,0),(0,2),(1,1),(2,0),(0,3),(1,2),(2,1),(3,0),(0,4),(1,3),(2,2),(3,1),(4,0)
    ,(0,5),(1,4),(2,3),(3,2),(4,1)]
```

ortogonálne

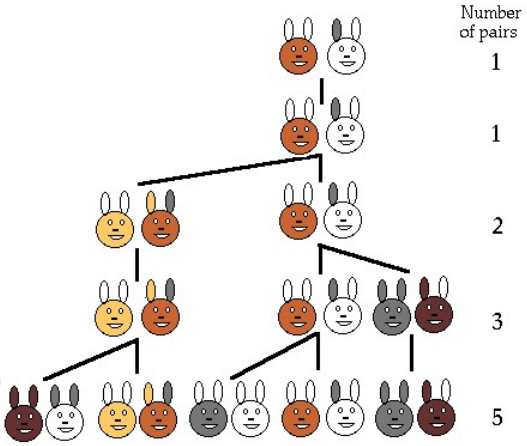
dvojice'' = foldr (++) []

[[(i,j) | j<-[0..i]] ++ [(j,i) | j<-[0..i-1]] | i<-[0..]]

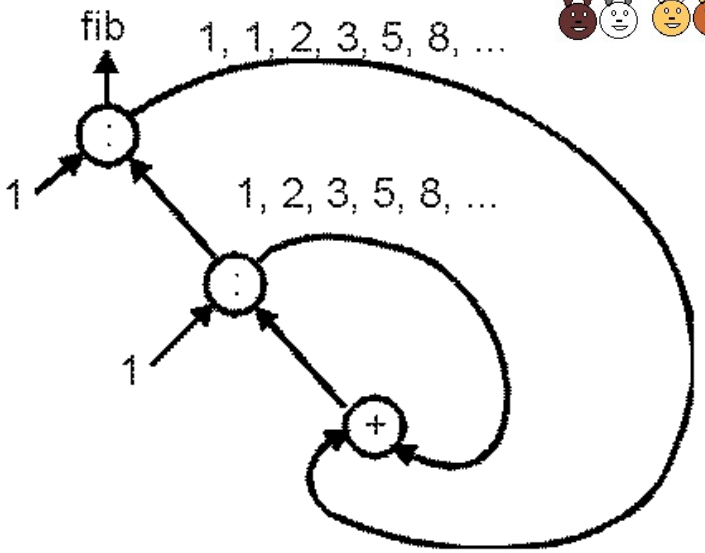



```
Main> take 20 dvojice''
[(0,0),(1,0),(1,1),(0,1),(2,0),(2,1),(2,2),(0,2),(1,2),(3,0),(3,1),(3,2),(3,3),(0,3),(1,3)
    ,(2,3),(4,0),(4,1),(4,2),(4,3)]
```

Fibonacciho zajace



fib 0 = 1
fib 1 = 1
fib (n+2) = fib (n+1) + fib n

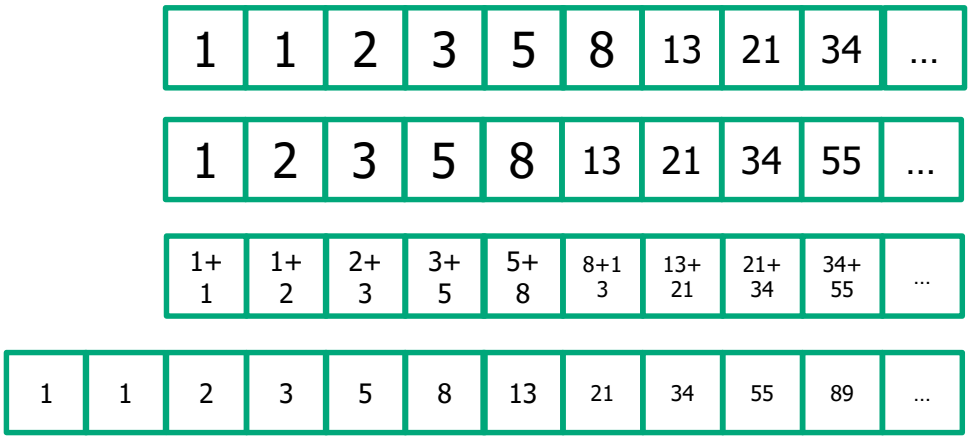



```
fib          = 1 : 1 : [ a+b | (a,b) <- zip fib (tail fib) ]

fib          = 1 : 1 : (map2 (+) fib (tail fib))
```

```
Main> take 7 fib
[1,1,2,3,5,8,13]
```

```
fibo@(_:tfib)    = 1 : 1 : [ a+b | (a,b) <- zip fibo tfib ]
```

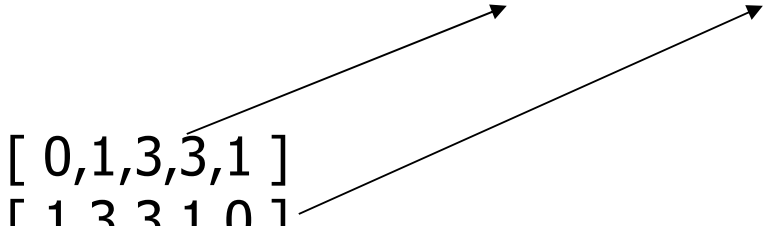



```
fib                    = 1 : 1 : [ a+b | (a,b) <- zip fib (tail fib) ]
```

Pascalov trojuholník

```
      1
    1   1
  1   2   1
1   3   3   1
1   4   6   4   1
```

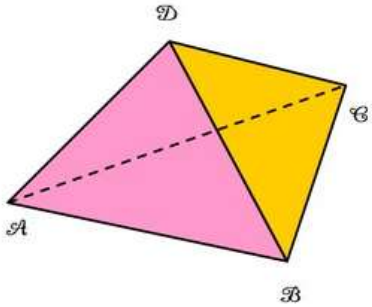
```
pascal :: [[Int]]
pascal = [1] : [[x+y | (x,y) <- zip ([0]++r) (r++[0])] | r <- pascal]
```



```
[ 0,1,3,3,1 ]                          [0]++r
[ 1,3,3,1,0 ]                          r++[0]
[ (0,1),(1,3),(3,3),(3,1),(1,0) ]      zip
[ 1,4,6,4,1 ]                          +
```

```
Main> take 5 pascal
[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

Pyramidové čísla



guličky daného počtu možno poskladať do pravidelného trojstenu - pyramidky:

f(x) platí, ak existuje n, že x = 1 + 3 + 6 + 10 + ... + (1 + ... + n)

Príklad, f(1), f(4), f(10), f(20) platí, ale f(5), f(21), f(34) neplatí.

Definujte nekonečný usporiadaný zoznam pyramida::[Int], ktorý obsahuje x, ak f(x) platí, take 5 f = [1,4,10,20,35].

```
pyramidy n       = sum [1..n]:pyramidy (n+1)
   -- pyramidy 1           = [1,3,6,10,15,21,...]
                              _
                              ___
                              _____
                              _______
pyramida = [ sum (take i (pyramidy 1)) | i<-[1..]]
```

Lenivé prvočísla
(Eratostenovo sito)

```
primes          :: [Int]
primes          = sieve [ 2.. ]
sieve (p:x)     = p : sieve [ n | n<-x, n `mod` p > 0 ]
```

```
sieve   [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,...] =
        2: sieve[3,5,7,9,11,13,15,17,19, ...] =
        2:3:sieve[5,7,11,13,17,19, ...] =
        2:3:5:sieve [7,11,13,17,19, ...] =
        2:3:5:7:sieve [11,13,17,19, ...] =
        2:3:5:7:11:sieve [13,17,19, ...] = ...
```

```
Main> take 10 primes
[2,3,5,7,11,13,17,19,23,29]
```

Lenivé prvočísla
(Eratostenovo sito)

- iná definícia pomocou iterate:

```
Main> take 10 (iterate (*2) 1)
[1,2,4,8,16,32,64,128,256,512]
```

```
iterate          :: (a -> a) -> a -> [a]
iterate f x      = x : iterate f (f x)
```

```
Main> take 10 primes'
[2,3,5,7,11,13,17,19,23,29]
```

```
primes'          :: [Int]
primes'          = map head (iterate sieve' [2 ..])

sieve'           :: [Int] -> [Int]
sieve' (p:ps)    = [x | x <- ps, x `mod` p > 0]
```

[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]
[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20, ...]

Hammingova postupnosť
(horská prémia)

Nech množina $M_{2,3,5}$ obsahuje 1 a s každým prvkom obsahuje jeho dvoj-, troj- a päťnásobok (v prémiovej úlohe aj sedemnásobok)

- ```
  hamming     :: [Integer]
  hamming      = 1 : (  map (2*) hamming ||
                        map (3*) hamming ||
                        map (5*) hamming)
            where (x:xs) || (y:ys)          -- klasický merge
                                | x==y  =  x : (xs || ys)
                                | x<y   =  x : (xs || (y:ys))
                                | y<x   =  y : (ys || (x:xs))
  ```

```
Main> take 100 hamming
[1,2,3,4,5,6,8,9,10,12,15,16,18,20,24,25,27,30,32,36,40,45,48,50,54,60,64,72,75,
80,81,90,96,100,108,120,125,128,135,144,150,160,162,180,192,200,216,225,240,243,
250,256,270,288,300,320,324,360,375,384,400,405,432,450,480,486,500,512,540,576,
600,625,640,648,675,720,729,750,768,800,810,864,900,960,972,1000,1024,1080,1125,
1152,1200,1215,1250,1280,1296,1350,1440,1458,1500,1536]
```
```

Horská prémia

M_2 = {2^a}
$M_{2,3}$ = {2^a*3^b}
$M_{2,3,5}$ = {2^a*3^b*5^c}
$M_{2,3,5,7}$ = {2^a*3^b*5^c*7^d }

- kolko2 x je počet prvkov množiny M_2 <= x
  kolko2 0=0
  kolko2 1=1
  kolko2 x=1+kolko2 (x `div` 2)

- kolko23 x je počet prvkov množiny $M_{2,3}$ <= x
- kolko23 0=0
  kolko23 x=kolko2 (x)+kolko23 (x`div` 3)
  a sú to tie, ktoré v rozklade nemajú 3 (b=0) plus tie, čo majú 3 (b>0)

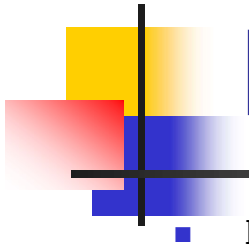
  kolko235 x je počet prvkov množiny $M_{2,3,5}$ <= x
- kolko235 0=0
  kolko235 x=kolko23 (x)+kolko235 (x`div` 5)
  a sú to tie, ktoré v rozklade nemajú 5 (c=0) plus tie, čo majú 5 (c>0)

- kolko2357 x je počet prvkov množiny $M_{2,3,5,7}$ <= x
  kolko2357 0=0
  kolko2357 x=kolko235 (x)+kolko2357(x`div` 7)
  a sú to tie, ktoré v prvočíselnom rozklade nemajú 7 plus tie, čo majú 7
  t.j. buď d = 0 alebo d > 0

Horská prémia

$M_{2,3,5,7} = \{2^a * 3^b * 5^c * 7^d\}$

```
Main> kolko2357 10                          = 10
Main> kolko2357 100                         = 46
Main> kolko2357 1000                        = 141
Main> kolko2357 10000                       = 338
Main> kolko2357 100000                      = 694
Main> kolko2357 1000000                     = 1273
Main> kolko2357 10000000                    = 2155
Main> kolko2357 100000000                   = 3427
Main> kolko2357 1000000000                  = 5194
Main> kolko2357 10000000000                 = 7575
Main> kolko2357 100000000000                = 10688
Main> kolko2357 1000000000000               = 19674
Main> kolko2357 10000000000000              = 42487
Main> kolko2357 100000000000000000          = 80988
Main> kolko2357 1000000000000000000000      = 141124
Main> kolko2357 100000000000000000000       = 118271
```

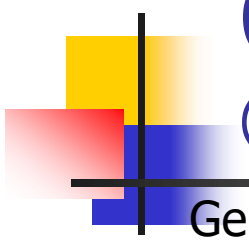


Horská prémia

- Main> hamming2357!!999
- 385875
- Main> hamming2357!!9999
- 63221760000
- Main> hamming2357!!99999
- 1230931449739687500000

- Main> find 1000
- [(385874,999),(385875,1000)]
- Main> find 10000
- [(63221759999,9999),(63221760000,Main>
- Main> find 100000
- [(1230931449739687499999,99999),(1230931449739687500000,100000)]

Python sa snaží byť lenivý
Ale skutočná lenivosť príde až Haskellom



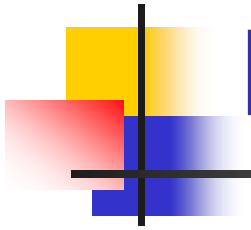
Generátory

(coroutiny)

Generátor je procedúra/funkcia, ktorá má v istom bode prerušený (a odložený) zvyšok svojho výpočtu.
Generátor odovzdáva výsledky volajúcej procedúre pomocou príkazu
    yield hodnota.
Na obnovenie výpočtu (a pokračovanie v ňom) generátora slúži funkcia
    next(gener)

```
def gen(n):                    # generátor generuje postupne čísla 0,1,2,...,n-1
    for i in range(n):         # cyklus pre i z intervalu
        yield i                # yield vždy preruší výpočet cyklu a urobí return i

print([x*x for x in gen(5)]) # for cyklus beží nad generátorom, [0,1,4,9,16]
print(sum(gen(5)))           # agregátor sum nad generátorom 10
print(list(gen(5))           # list nad generátorom pozbiera jeho výsledky
g = gen(5)
print(next(g)),print(next(g)),print(next(g)),print(next(g)),print(next(g)),print(nex
0              1              2              3              4              Exception
```



Nekonečné generátory

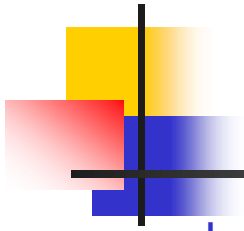
```
def integers(n):                # generuje nekonečne veľa výsledkov tvaru
    while True:                 # n, n+1, n+2, …
        yield n
        n += 1

print(list(integers(1)))        # toto nemôže nikdy vytvoriť celý zoznam
print(min(integers(1)))         # hoc minimum je zrejmé, ani toto nedobehne
[n*2 for n in integers(1)]      # tu by už Haskell niečo dal, ale Python nie ...

def take(n,g):                  # zober prvých n generovaných hodnôt gen. g
    for i in range(n):
        yield next(g)           # next(g) vyprovokuje výpočet ďalšej hodnoty g

print(list(take(10,integers(1))))       # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

s nekonečnom sa dá pracovať len lenivo



Eratosten

```
def sieve(d, sieved):                  # osievací generátor
    for x in sieved:                   # z generátora sieved prepúšťa len
        if (x % d != 0):               # hodnoty nedeliteľné d
            yield x

def eratosten(ints):                   # eratostenovo sito (prvočísla :-)
    while True:                        # zober generátor ints=integers(2)
        first = next(ints)             # prvé číslo predstavuje prvočíslo
        yield first                    # toto sa reportuje výsledok eratosten
        ints = sieve(first, ints)      # preosejeme čísla tak, že vyhádžeme
                                       # všetky deliteľné týmto prvočíslom
                                       # a pokračujeme v cykle

print(list(take(100,eratosten(integers(2)))))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113,
127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383,
389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541]
```