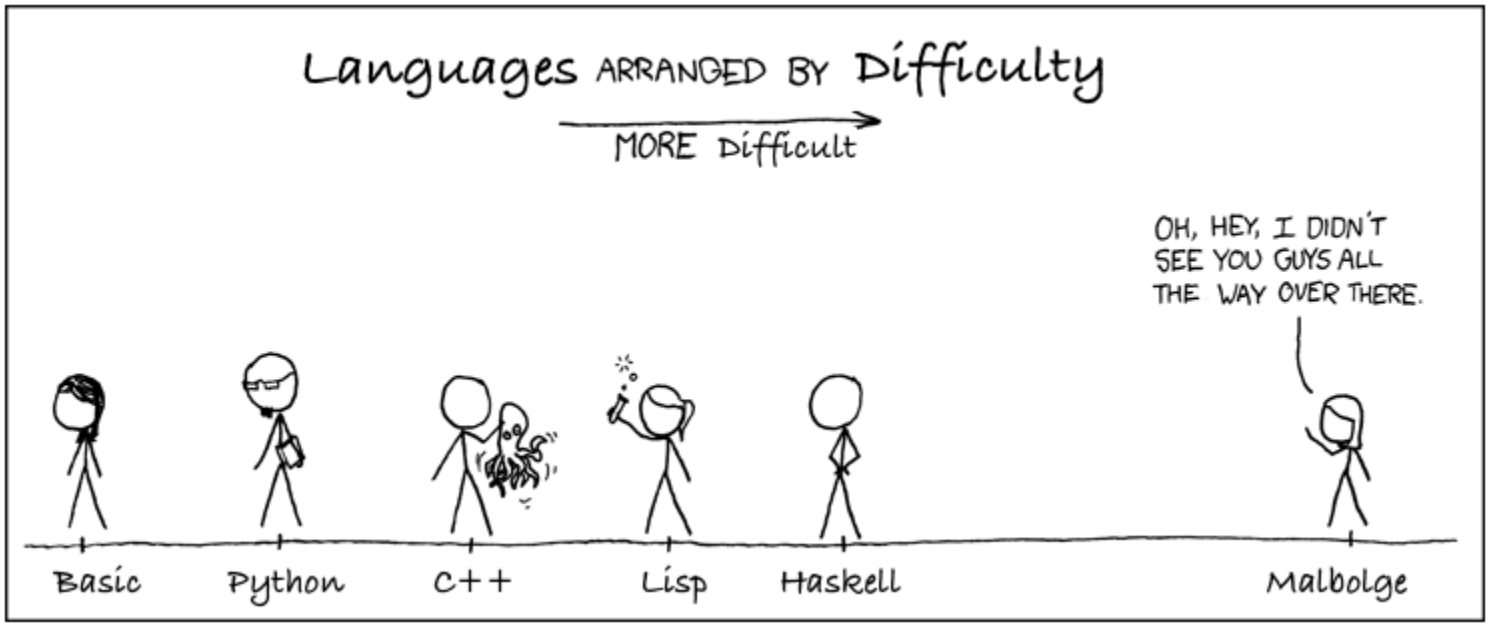
HaskellThon



https://en.wikipedia.org/wiki/Malbolge

https://medium.freecodecamp.org/what-programming-language-should-i-learn-first-%CA%87d%C4%B1%C9%B9%C9%94s%C9%90%CA%8C%C9%90%C9%BE-%C9%B9%C7%9D%CA%8Dsu%C9%90-19a33b0a467d

import Data.List

http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-List.html

Data.List

Operations on lists.

Copyright	(c) The University of Glasg
License	BSD-style (see the file libra
Maintainer	libraries@haskell.org
Stability	stable
Portability	portable
Safe Haskell	Trustworthy
Language	Haskell2010

Basic functions

```
(++) :: [a] -> [a] -> [a]    infixr 5    # Source
```

Append two lists, i.e.,

```
[x1, ..., xm] ++ [y1, ..., yn] == [x1, ..., xm, y1, ..., yn]
[x1, ..., xm] ++ [y1, ...] == [x1, ..., xm, y1, ...]
```

If the first list is not finite, the result is the first list.

```
head :: [a] -> a    # Source
```

Extract the first element of a list, which must be non-empty.

```
last :: [a] -> a    # Source
```

Extract the last element of a list, which must be finite and non-empty.

```
tail :: [a] -> [a]    # Source
```

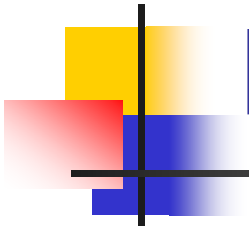
Extract the elements after the head of a list, which must be non-empty.

```
init :: [a] -> [a]    # Source
```

Return all the elements of a list except the last one. The list must be non-empty.

Contents

Všetko,čo by ste chceli vediet o Haskelli, ale báli ste sa spýtať



- že **a b c d = (((a b) c) d)**…lebo operátor aplikácie funkcie na argument je *ľavo asociatívny*, teda ak zabudnem zátvorky, tak ich chápe doľava
- **Int -> Int -> Int -> Char = Int -> (Int -> (Int -> Char))** … lebo operátor funkčného typu **->** je *pravo asociatívny*, teda ak zabudnem zátvorky, tak ich chápe doprava. Explicitne, (Int->Int) -> (Int -> Int)
- **Int -> Int -> String != (Int, Int) -> String**… lebo prvé je funkcia, ktorá vráti funkciu, ktorá vráti String. Vďaka *currying* ju volám takto f 4 5, čo je (f 4) 5. Druhé je funkcia, ktorá čaká dvojicu. Musím ju volať takto: g (4,5), a vyzerám, že som Javista, a na Haskelli prvý týždeň...
- **Int != Integer** … lebo Int z interval minBound::Int ... maxBound::Int =9223372036854775807=2^63-1, ergo to je **long**. Integer je BigInteger
- ako sa konvertuje **Int, Integer, Float** … to neviem ani ja, googlim…

Všetko,čo potrebujem vedieť, ma mali naučiť v materskej škôlke

- **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

Bojím sa spýtať, čo všetko ma nenaučili v materskej škôlke...

Na typoch záleží (aj keď 'detstvo' bez nich bolo krásne a jednoduché):

- [[t]] nikdy nebude [t] (List<List<Integer>> nie je List<Integer>)

preto nemôžem napísať

- [ch+(slova k) | ch <- "ABCDEF"]

```
>:type "ABCDEF"
"ABCDEF" :: [Char]
```

slova k :: [String] == [[Char]], … lebo **type** String = [Char]

ch+(slova k) znamená Char + [[Char]]

Okrem toho, zreťazenie zoznamov je (**++**) :: [t] -> [t] -> [t]
Ale ani ch++(slova k) nie je dobre, lebo je to Char ++ [[Char]], nepasuje…

Prilep ako hlavu k zoznamu je (**:**) :: t -> [t] -> [t]
Ale ani ch : (slova k) nie je dobre, lebo je to Char : [[Char]], nepasuje…

Píšte (si) typy (kdekoľvek sa dá)**, sú zdravé**, a hlášky GHC potom čitateľnejšie

Slova, která jsem si přál napsat sám – Robert Fulghum

```
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","FI
E","FE","AF","BF","CF","DF","EF","FF"]

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

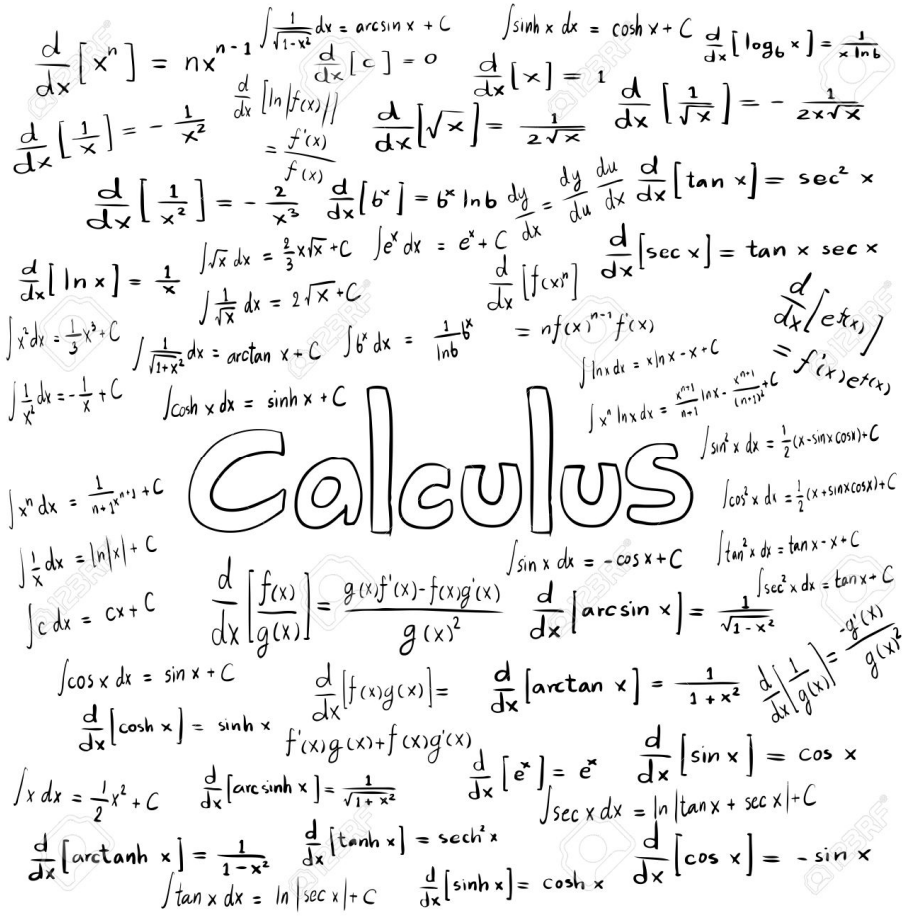
length $ nub $ slova' 2 = 43

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



WolframAlpha

(6^(k+1)-1)/5, k in 1..10

Input:

$$\text{Table}\left[\frac{1}{5}\left(6^{k+1}-1\right), \{k, 1, 10\}\right]$$

Result:

k	$\frac{1}{5}\left(6^{k+1}-1\right)$
1	7
2	43
3	259
4	1555
5	9331
6	55 987
7	335 923
8	2 015 539
9	12 093 235
10	72 559 411

https://www.ted.com/talks/stephen_wolfram_computing_a_theory_of_everything

Všetko, čo ste chceli zmeniť, a nikdy sa vám to nepodarilo

- zoznam ("pole") xs vieme indexovať indexami i <- [0..length xs-1]
  xs!!i -- getter
- neexistuje setter xs[i] = value

```
set  ::  [t] -> Int -> t -> [t]
set xs i value | i < 0            = xs    -- out of range
               | i >= length xs   = xs    -- out of range
               | otherwise        = (if i == 0 then value else y):set ys (i-1) value
                                    where (y:ys) = xs
               | otherwise        = let (y:ys) = xs in
                                    (if i == 0 then value else y):set' ys (i-1) value
set''  ::  [t] -> Int -> t -> [t]
set'' xs i value | i < 0           = xs    -- out of range
                 | i >= length xs  = xs    -- out of range
                 | otherwise       = [xs!!j | j <- [0.. i-1] ] ++ [value] ++
                                     [xs!!j | j <- [i+1..length xs-1] ]
```

zoznam.hs

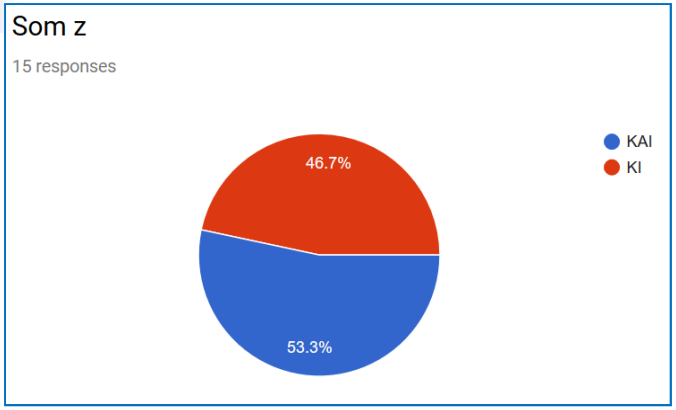
Haskell homework tu nevidím...

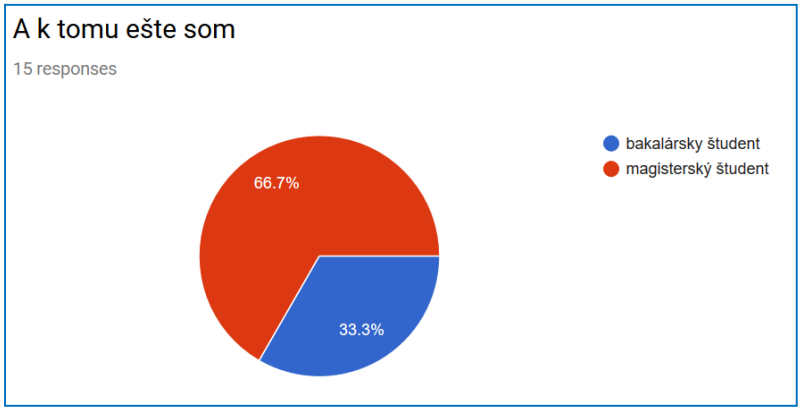
- Share everything.
- Play fair.
- Don't hit people.
- Put things back where you found them.
- Clean up your own mess.
- Don't take things that aren't yours.
- Say you're sorry when you hurt somebody.
- Wash your hands before you eat.
- Flush.
- Warm cookies and cold milk are good for you.
- Live a balanced life—learn some and think some and draw and paint and sing and dance and play and work every day some.
- Take a nap every afternoon.

https://sk.wikipedia.org/wiki/Robert_Fulghum

Anketa

(107%)

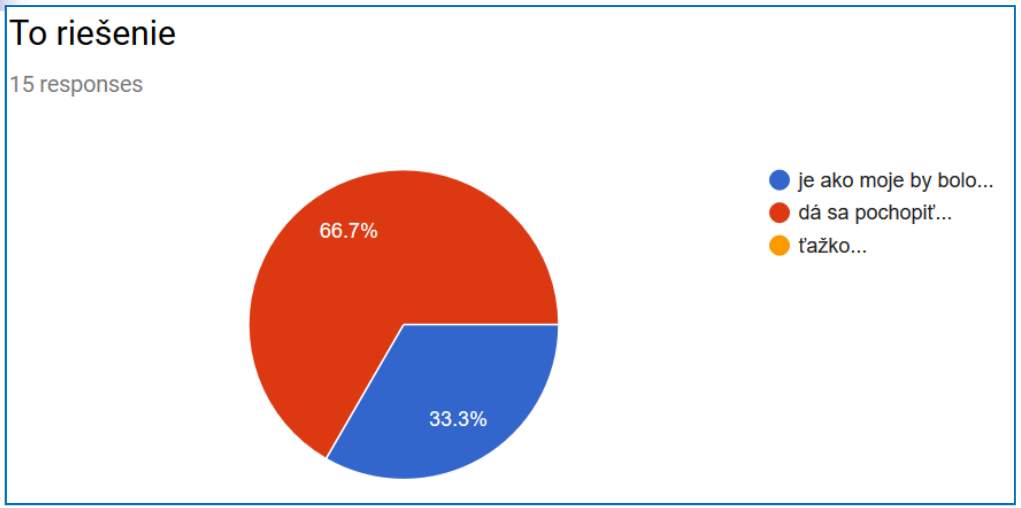

Som z
15 responses
46.7%
53.3%
KAI
KI


A k tomu ešte som
15 responses
66.7%
33.3%
bakalársky študent
magisterský študent


Ktorému zadaniu najlepšie rozumiete?
15 responses
93.3%
slová nad abecedou písmen A,B,C,D, E,F dĺžky k
k-prvkové variácie množiny {A,B,C,D, E,F}
{A,B,C,D,E,F}^k

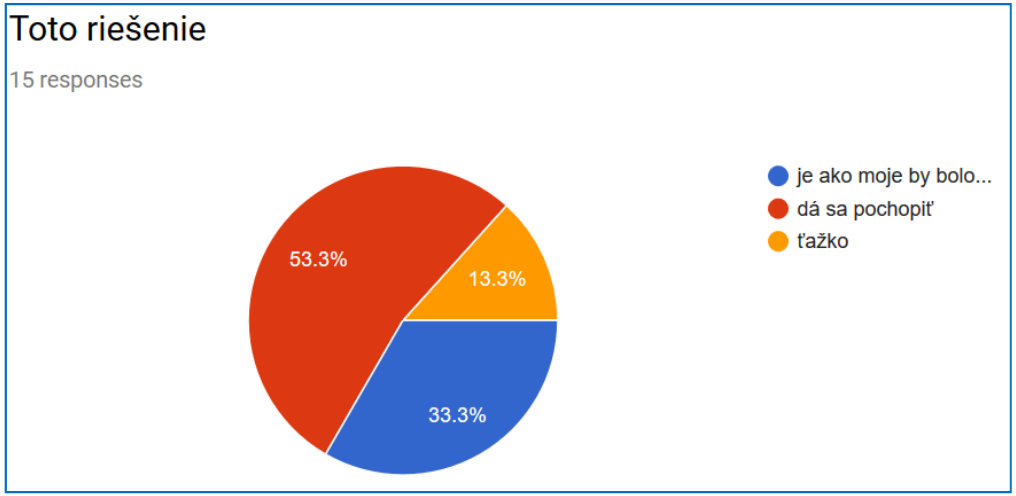
Anketa

(107%)




```
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))
```

Anketa
(107%)




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

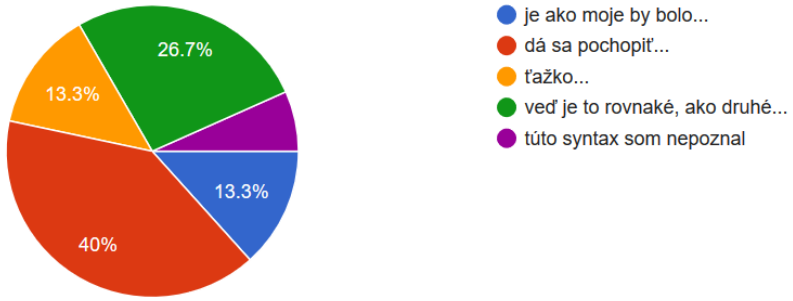
Anketa

(107%)

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

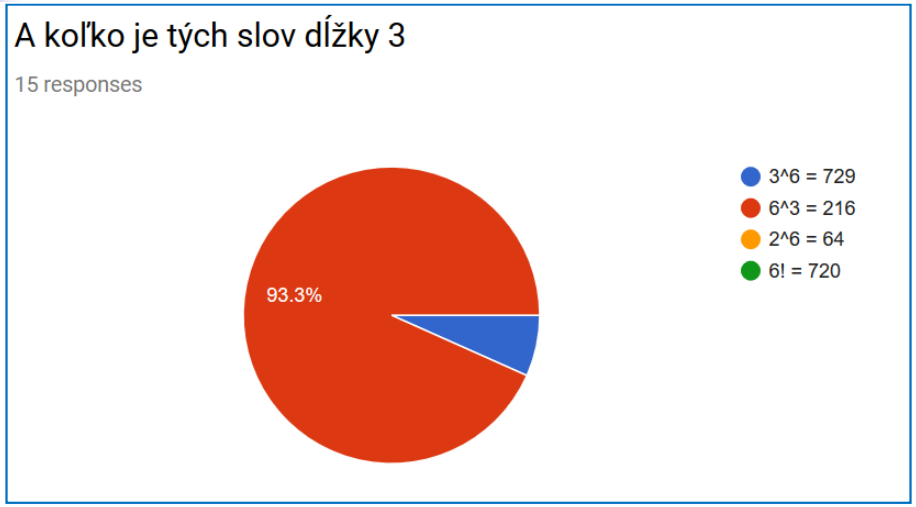
Toto riešenie

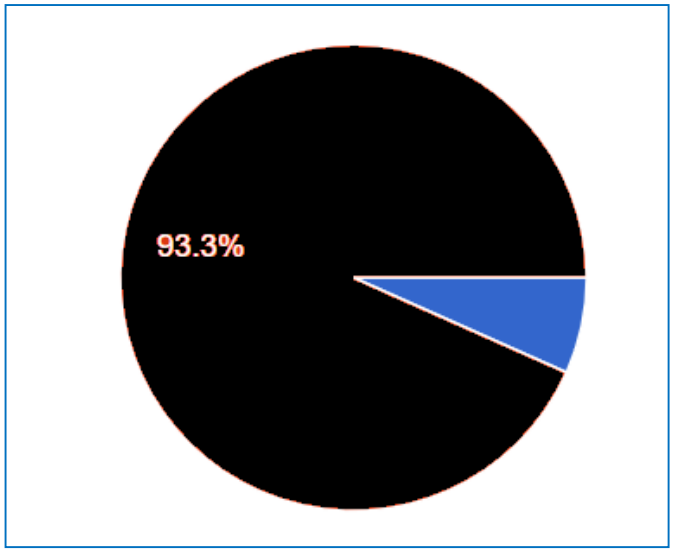
15 responses

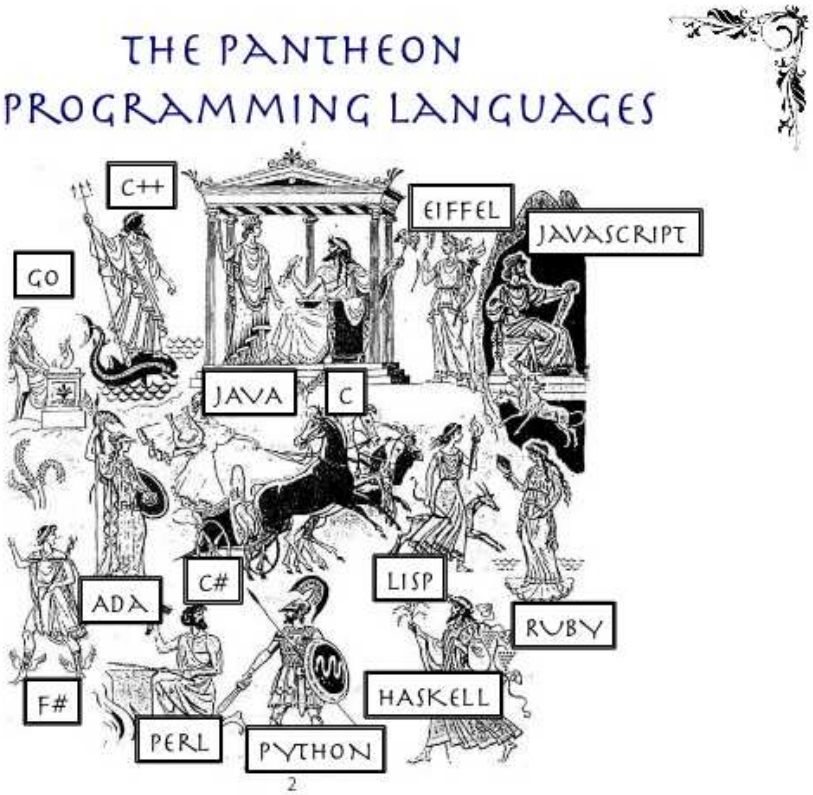



```
def words(len):
        if len == 0:
                return [""]
        else:
                return (ch+slovo
                        for slovo in words(len-1)
                        for ch in 'ABCDEF')
for m in words(3):
        print(m)
```

Anketa / Rozcvička 1







- .js
  - array-comprehension
  - `[for (x of iterable) if (condition) x]`
  - `var numbers = [1, 2, 3, 21, 22, 30];`
    `[for (i of numbers) if (i % 2 === 0) i];`
  - iterátor/generátor
- .py
  - async/await (coroutines)
  - destructor (destructor assignment)

```
                                    def foo(): return [1,2,3,4]
  (x, *xs) = [1,2,3,4]              (x, *xs) = foo()      x = 1, xs = [2,3,4]
  (x, y, *ys) = [1,2,3,4]           (x, y, *ys) = foo()   x = 1, y = 2, ys = [3,4]
```

  - čo ak foo() je generátor ?
- .hs
  - lazy evaluation (generátory)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Iterators_and_Generators
https://docs.python.org/3/library/asyncio-task.html

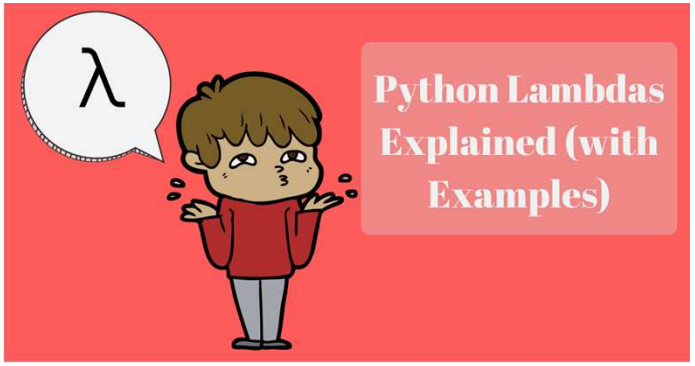
All Things Pythonic

Guido van Rossum: The fate of reduce() in Python 3000, (r.2005)

- Python aquired lambdas, reduce(), filter() and map() thanks a Lisp hacker
- despite of the PR value, I think these features should be cut from Python 3
- **Update: lambda, filter and map will stay (the latter two with small changes, returning iterators instead of lists). Only reduce will be removed from the 3.0 standard library. You can import it from functools.**

https://www.artima.com/weblogs/viewpost.jsp?thread=98196

Python Kvíz



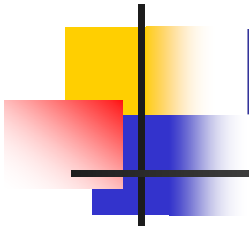

```
print(map(lambda x: x*x, [1,2,3,4,5]))                          <map object at 0x037
print(list(map(lambda x: x*x, [1,2,3,4,5])))                    [1, 4, 9, 16, 25]
print(list(filter(lambda y:y>10,map(lambda x: x*x, [1,2,3,4,5]))))
                                                                [16, 25]

from functools import reduce
print(reduce((lambda x, y: x * y), [1, 2, 3, 4]))               24

print(reduce((lambda x, y: x + y), [1, 2, 3, 4]))               10
print(reduce((lambda x, y: x - y), [1, 2, 3, 4]))               -8

def compose(f, g):
        return lambda x: f(g(x))
print(compose( lambda x: x+1, lambda x: x*3 )(10))              31

def composeMany(*fs):
        return reduce(compose, fs)                              33
print(composeMany(lambda x:x+1, lambda x:x+2, lambda x:x*3)(10))
```

lambdas.hs

Does not matter much…

for job: Better choice would be Scala (modern Java)
https://www.coursera.org/learn/progfun1

for school: Haskell



List-comprehension

Každý poriadny kurz FP začína funkcionálmi map a filter:

...ale my sme trénovali list-comprehension:

[f x | x <- xs, p x] [f(x) for x in xs if p(x)]

```
map          ::   (a -> b) -> [a] -> [b]
map f xs     =    [ f x | x <- xs]

filter       ::   (a -> Bool) -> [a] -> [a]
filter p xs  =    [ x | x <- xs, p x]
```