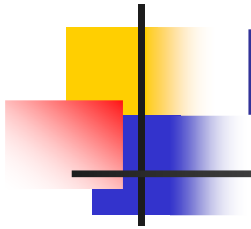
The Evolution of a Haskell Programmer



Inšpirovaný rôznymi technikami programovania v Haskelli riešime nasledujúce zadania:

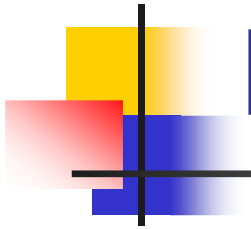
fac n = 1.2. ... .n

sum n = 1+2+ ... +n

sumsq n = 1^2 + 2^2 + ... + n^2

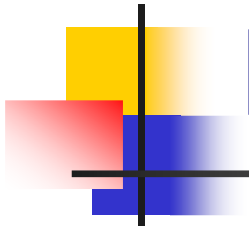
triangle n = [[1,2,..n],[1,2,..,n-1],..,[1,2],[1]]

http://www.willamette.edu/~fruehr/haskell/evolution.html



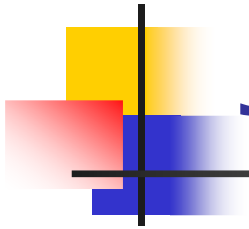
Freshman Haskell programmer

```
fac n = if n == 0 then 1 else n * fac (n-1)

suma n = if n == 0 then 0 else n + suma (n-1)

sumsq n = if n == 0 then 0 else (n*n) + sumsq (n-1)

triangle n = if n == 0 then [] else [1..n] : triangle (n-1)
```



Haskell programmer, at MIT

```
fac1 =
  (\(n) -> (if ((==) n 0) then 1 else ((*) n (fac1 ((-) n 1)))))

suma1 =
  (\(n) -> (if ((==) n 0) then 0 else ((+) n (suma1 ((-) n 1)))))

sumsq1 =
   (\(n) -> (if ((==) n 0) then 0 else ((+) ((*) n n) (sumsq1 ((-) n 1)))))

triangle1 =
   (\(n) -> (if ((==) n 0) then [] else (:) [1..n] (triangle ((-) n 1))))
```

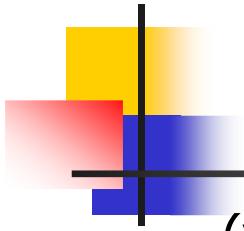


Junior Haskell programmer

```
fac2 0                  = 1
fac2 (n+1)              = (n+1) * fac2 n

sum2 0                  = 0
sum2 (n+1)              = (n+1) + sum2 n


sumsq2 0                = 0
sumsq2 (n+1)            = (n+1)*(n+1) + sumsq2 n

triangle2 0             = []
triangle2 (n+1)         = [1..n] : triangle2 n
```

- **Another junior Haskell programmer**

```
fac 0     = 1
fac n     = n * fac (n-1)
```

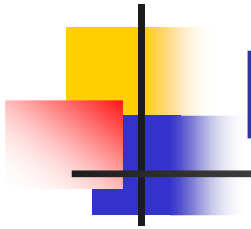


Senior Haskell programmer

(voted for John McCain — “leans right”)

```
fac3 n = foldr (*) 1 [1..n]

sum3 n = foldr (+) 0 [1..n]

sumsq3 n   = foldr (+) 0 (map (\x->x*x) [1..n])
sumsq33 n = foldr (\x y->x*x+y) 0 [1..n]

triangle3 n =
    foldr (\x y->[1..x]:y) []
          (enumFromThenTo n (n-1) 1)
```

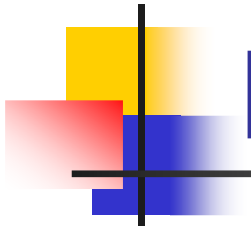
Another senior Haskell programmer



(voted for McGovern Barack Obama — "leans left")

```
fac4 n = foldl (*) 1 [1..n]

sum4 n = foldl (+) 0 [1..n]
sumsq4 n = foldl (\x y->y*y+x) 0 [1..n]

triangle4 n = foldl (\x y->[1..y]:x) [] [1..n]
```

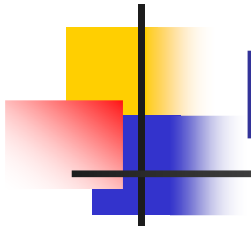
Yet another senior Haskell programmer



(leaned so far right he came back left again!)

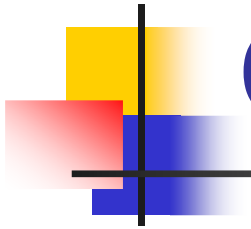
```
-- using foldr to simulate foldl

fac5 n = foldr (\x g n -> g (x*n)) id [1..n] 1

sum5 n = foldr (\x g n -> g (x+n)) id [1..n] 0

sumsq5 n = foldr (\x g n -> g (x*x+n)) id [1..n] 0

triangle5 n = foldr (\x g n -> g ([1..x]:n)) id [1..n] []
```

Memoizing Haskell programmer



```
fac6 n = facs !! n
facs = scanl (*) 1 [1..]

sum6 n = sums !! n
sums = scanl (+) 0 [1..]

sumsq6 n = sumsqs !! n
sumsqs = scanl (\x y -> y*y+x) 0 [1..]

triangle6 n = triangles !! n
triangles = scanl (\x y -> [1..y]:x) [] [1..]

scanl :: (a -> b -> a) -> a -> [b] -> [a]
scanl f q xs = q : (case xs of   [] -> []
                                 x:xs -> scanl f (f q x) xs)
```

```
scanl (*) 1 [1..] !! 10
```

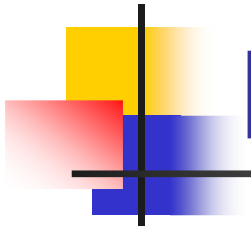


Oxford Haskell programmer

```
fac = foldr (*) 1 . enumFromTo 1

fac7 = foldr (*) 1 . enumFromTo 1

sum7 = foldr (+) 0 . enumFromTo 1

sumsq7 = foldr (+) 0 . map (\x->x*x) . enumFromTo 1

triangle7 = foldr (:) [] . map (\x->[1..x]) . enumFromTo 1
```

```
(foldr (*) 1 . enumFromTo 1) 10
(foldr (*) 1 . enumFromTo 1) 10
foldr (*) 1 (enumFromTo 1 10)
```

Accumulating Haskell programmer

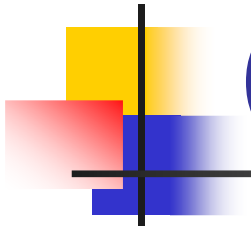


```
facAcc a 0      = a
facAcc a n      = facAcc (n*a) (n-1)
fac8            = facAcc 1

sumAcc a 0      = a
sumAcc a n      = sumAcc (n+a) (n-1)
sum8            = sumAcc 0

sumsqAcc a 0    = a
sumsqAcc a n    = sumsqAcc (n*n+a) (n-1)
sumsq8          = sumsqAcc 0

triangleAcc a 0 = a
triangleAcc a n = triangleAcc ([1..n]:a) (n-1)
triangle8       = triangleAcc []
```

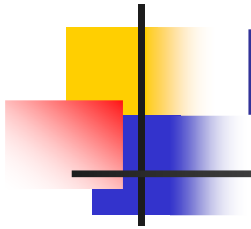
Iterative Haskell programmer (Pascal programmer)

```
fac n = result (for init next done)
        where      init = (0,1)
                   next   (i,m) = (i+1, m * (i+1))
                   done   (i,_) = i==n
                   result (_,m) = m

for i n d = until d n i

until :: (a -> Bool) -> (a -> a) -> a -> a
until p f x = if p x then x else until p f (f x)

triangle9 n = result (for init next done)
        where init = (0,[])
              next   (i,m) = (i+1, [1..(i+1)]:m)
              done   (i,_) = i==n
              result (_,m) = m
```

Iterative one-liner Haskell (APL or C) programmer

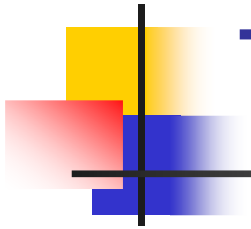


```
fac10 n =
    snd (until ((>n) . fst) (\(i,m) -> (i+1, i*m)) (1,1))

sumsq10 n =
    snd (until ((>n) . fst) (\(i,m) -> (i+1, i*i+m)) (1,0))

triangle10 n =
    snd (until ((>n) . fst) (\(i,m) -> (i+1, [1..i]:m)) (1,[]))
```



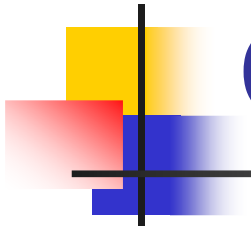
Boy Scout Haskell programmer

```
y f       = f (y f)
fac11     = y (\f n -> if (n==0) then 1 else n * f (n-1))

sumsq11           = y (\f n -> if (n==0) then 0 else n*n + f (n-1))
```



Tenured professor

```
fac12 n = product [1..n]

sumsq12 n = n*(n+1)*(2*n+1) `div` 6

triangle11 n = map (\x -> [1..x]) (enumFromThenTo n (n-1) 1)
```



Cvičenie

Rôznymi spôsobmi definujte funkciu, ktorej výsledkom je jednotková matica rozmeru n

- jednotkova n = [[1,0,...,0],[0,1,...,0],..,[0,...,1,0],[0,...,0,1]]

Definujte rôzne algoritmy pre triedenie zoznamu:

- bubleSort
- insertSort
- selectionSort