Pointfree style

```
lcaseString s = map toLower s
lcaseString   = map toLower

sum' xs = foldr (+) 0 xs

lcaseLetters s = map toLower (filter isAlpha s)
lcaseLetters s = (map toLower . filter isAlpha) s
lcaseLetters   = map toLower . filter isAlpha

fstGt0 l = filter (\ (a,b) -> a>0) l
fstGt0   = filter (\ (a,b) -> a>0)
fstGt0   = filter (\x -> fst x > 0)
fstGt0   = filter (\x -> ((>0) . fst) x)
fstGt0   = filter ((>0).fst)

pair x = (x, x)
square x = x * x
square   = uncurry (*) . pair
```

♣ K vyzkoušení

```
func1 x l = map (\y -> y * x) l

func2 f g l = filter f (map g l)

func3 f l = l ++ map f l

func4 l = map (\y -> y+2)
              (filter (\z -> z `elem` [1..10])
                      (5:l))

func5 f l = foldr (\x y -> f (y,x)) 0 l
```

♣ Ukázka řešení func2

```
func2 f g l = filter f (map g l)
func2 f g = (filter f) . (map g)         -- definition of (.)
func2 f g = ((.) (filter f)) (map g)     -- desugaring
func2 f = ((.) (filter f)) . map         -- definition of (.)
func2 f = flip (.) map ((.) (filter f)) -- desugaring, def. of flip
func2 = flip (.) map . (.) . filter      -- def. of (.), twice
func2 = (. map) . (.) . filter           -- add back some sugar
```

♣ Ukázky nástroje pointfree

```
\x y -> (x, y)                          = (,)
\x y -> (f x, y)                        = (,) . f
\x y -> (x, g y)                        = (. g) . (,)
\x y -> (f x, g y)                      = (. g) . (,) . f

\x -> x * x                             = join (*)
join (*) :: (Num a, Monad ((->) a)) => a -> a

ap :: Monad m => m (a -> b) -> m a -> m b
\x -> (x, f x)                          = (,) `ap` f
(,) `ap` (const undefined) :: Monad ((->) a1) => a1 -> (a1, a)
```