1. HS1 - súdeliteľné

domaca.hs

```
sudelitelne :: Int -> Int -> Bool
sudelitelne a b = gcd a b > 1
```

Klobúk dole Haskellu, že už v základe obsahuje funkcie ako *gcd*. Robí ma to neuveriteľne šťastným. Tým pádom je funkcia vyššie asi zrejmá bez vysvetľovania a vychádza priamo z definície v zadaní: *Čísla sú súdeliteľné, ak ich najväčší spoločný deliteľ je viac ako 1.*

2. HS1 - NnadK

domaca.hs

```
comb :: Int -> Int -> Int
comb n k = product [1..n] `div` (product [1..k] * product [1..n-k])

-- potom zoznam 'rows' riadkov Pascalovho trojuholnika
pascal rows = [[comb n k | k <- [0..n]] | n <- [0..rows]]
```

Na výpočet faktoriálu som využil vstavanú funkciu *product* a išiel podľa známeho vzorca $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Následne som si vyskúšal vzhľadom na dnešné cviko (5.11) *list comprehension* a vygeneroval si tak *rows* prvých riadkov Pascalovho trojuholníka.

3. HS1 - geom

domaca.hs

```
jeGeom :: [Int] -> Bool
jeGeom xs | length xs < 3 = True
jeGeom (x:y:z:xs) = (x * q == y) && (y * q == z) && jeGeom (y:z:xs)
       where q = y `div` x
```

Rekurzívne kontrolujeme trojice prvkov x, y, z, tak, že porovnáme na rovnosť výrazy xq a y, yq a z. Koeficient q sa dopočíta ako podiel y a z.

4. HS1 - jePrefix

domaca.hs

```
jePrefix :: [Int] -> [Int] -> Bool
jePrefix [] xs = True
jePrefix (x:xs) [] = False
jePrefix (x:xs) (y:ys) = x == y && jePrefix xs ys
```

Rekurzívne kontrolujme postupne rovnosť dvojice prvkov na začiatku oboch zoznamov, kým je prvý (alebo oba) neprázdny.

5. HS1 - int2Bin

domaca.hs

```
int2Bin :: Int -> [Int]
int2Bin 0 = [0]
int2Bin 1 = [1]
int2Bin n | n `mod` 2 == 0 = int2Bin (n `div` 2) ++ [0]
          | otherwise = int2Bin (n `div` 2) ++ [1]
```

V tejto funkcii simulujem klasický 'papierový' proces prevodu desiatkového čísla na binárne. Delíme číslo n dvojkou kým sa dá a popritom si pamätáme zvyšok. Pri vynáraní z rekurzie tento zvyšok vo formáte jednoprvkového zoznamu reťazíme s medzivýsledkom. Po vynorení máme z desiatkového čísla binárne v správnom poradí.

6. HS1 - medián

domaca.hs

```
median :: [Float] -> Float
median [] = 0
median xs = result
       where n = length xs
             midIdx = n `div` 2 -- or (n-1) `div` 2
             result = sort xs !! midIdx
```

Za využitia premenných v časti *where* si najprv vypočítam stredný index zoznamu - buď $(n-1)$ // 2 alebo n // 2 pre výber ľavého, resp. pravého stredného prvku pri $n\%2 = 0$. Do výsledku priradím prvok na tomto strednom indexe v **usporiadanom** zozname.

7. HS1 - rotácia

domaca.hs

```
rotuj :: Int -> [a] -> [a]
rotuj 0 xs = xs
rotuj n xs = rotuj (n-1) (last xs : init xs)
```

Neviem či najefektívnejšie riešenie (zrejme nie), ale pomerne elegantné. Pri vynáraní z rekurzie sa posledný prvok zoznamu n krát presunie na začiatok a zvyšných $n-1$ prvkov sa spolu pomocou *init* posúva nakoniec.

8. HS1 - tretí

domaca.hs

```
treti :: [Int] -> Int
treti xs | length xs < 3 = error "List length must be at least 3"
         | otherwise = sort xs !! (length xs - 3)
```

Pri zozname dĺžky menšej ako 3 vyhodím výnimku. Inak bez použitia náročného *reverse* vyberiem zo **vzostupne usporiadaného** zoznamu tretí prvok od konca.

9. HS1 - prienik

domaca.hs

```
prienik :: [Int] -> [Int] -> [Int]
prienik xs ys = nub [x | x <- xs, x `elem` ys]
```

Úplne jednoducho inšpirujúc sa matematickou definíciou, vyberme také x zo zoznamu xs, kde x sa nachádza v ys. Tento výsledný zoznam ešte prežeňme funkciou *nub*, ktorá z neho vyfiltruje unikátne hodnoty.