1. [povinné][Haskell] Naprogramujte funkci `myZip :: [a] → [b] → [(a,b)]`, která se chová stejně jako její obdoba ze standardní knihovny `zip`. Tato funkce spojí zadané dva seznamy tak, že výsledný seznam bude na každé pozici obsahovat dvojici, která se skládá z prvků na odpovídajících pozicích v prvním a ve druhém zadaném seznamu. Pokud se délky původních seznamů liší, výsledný seznam bude mít délku kratšího z nich. Můžete využít libovolné konstrukce jazyka Haskell, nesmíte však použít knihovní funkce `zip` a `zipWith`.

Příklady vyhodnocení:

```
myZip [3, 2, 1] [1, 2, 3] ⇝* [(3,1), (2,2), (1,3)]
myZip [3, 2, 1] ['a', 'b'] ⇝* [(3,'a'), (2,'b')]
myZip [] ['a', 'b'] ⇝* []
```

---

2. [povinné][Haskell] Uveďte typ výrazu `zipWith (<)`. Typy elementárních výrazů jsou:

```
zipWith :: (a → b → c) → [a] → [b] → [c]
(<) :: Ord a ⇒ a → a → Bool
```

3. [povinné][Haskell] Mějme dány datové typy `Point` a `LPicture`, jejichž hodnoty slouží k popisu dvojrozměrného vektorového obrázku složeného z jednoduchých čar. (Každá čára je popsána dvěma krajními body, každý bod je určen souřadnicemi x a y, tj. dvěma celými čísly.) Obrázek je tvořen buď samostatnou čárou, nebo kompozicí dvou obrázků.

```
data Point = P Int Int
data LPicture = Line Point Point | Comp LPicture LPicture
```

a) Uveďte libovolnou platnou hodnotu typu `LPicture`, pro jejíž konstrukci je použit datový konstruktor `Comp`.

b) Napište funkci `flipXY :: LPicture → LPicture`, která prohodí hodnoty souřadnic x a y ve všech bodech použitých pro popis obrázku.

4. [povinné][Prolog] Zakreslete kompletní SLD strom pro dotaz `?- d(Y, Y), d(Y, a).` a databázi faktů níže. Pro úspěšné větve zapište výslednou substituci, kterou by interpretr vypsal na obrazovku.

```
f(a).
f(c).
d(X, Y) :- f(X), f(Y).
```

5. [povinné][Prolog] Naprogramujte predikát `consecutive(+List)`, který uspěje právě tehdy, když seznam `List` obsahuje alespoň jednu dvojici bezprostředně po sobě jdoucích prvků, ze kterých druhý je o jedničku větší než první. Pokud vstupní seznam obsahuje takových dvojic více, může predikát uspět vícekrát, ale rozhodně musí uspět alespoň jednou. Můžete předpokládat, že `List` je plně instanciovaný seznam čísel.

Příklady vyhodnocení:

```
?- consecutive([1, 5, 3, 2]).
false.

?- consecutive([4, 2, 3]).
true.

?- consecutive([]).
false.
```