Übungsblatt07 ; Thomas Alessandro Buse ; 192959 ; Gruppe: 17 ; Paul Rüssmann ; 196683

```
--Aufgabe 6.1
data Nat = Zero | Succ Nat

--Aufgabe 6.1 a
instance Eq Nat where
        (==) Zero Zero = True
        (==) _ Zero = False
        (==) Zero _ = False
        (==) (Succ a) (Succ b) = (==) a b

--Aufgabe 6.1 b
instance Ord Nat where
        (<=) Zero Zero = True
        (<=) _ Zero = False
        (<=) Zero _ = True
        (<=) (Succ a) (Succ b) = (<=) a b

--Aufgabe 6.1 c
instance Enum Nat where
        toEnum 0 = Zero
        toEnum n = Succ(toEnum(n-1))

        fromEnum Zero = 0
        fromEnum (Succ n) = 1 + fromEnum n

--Aufgabe 6.1 d
instance Show Nat where
        show Zero = "0"
        show (Succ n) = show(read (show(n)) + 1)
```

```
--Aufgabe 6.1 e

instance Num Nat where

        negate = undefined

        abs n = n

        signum Zero = Zero

        signum n = Succ Zero

        fromInteger = toEnum . fromInteger


        (+) a Zero = a

        (+) Zero b = b

        (+) a (Succ b) = (Succ a) + b


        (*) a Zero = Zero

        (*) Zero b = Zero

        (*) a (Succ b) = a + (a * (b))


--Aufgabe 6.1 f

solutions :: [(Nat,Nat,Nat)]

solutions = [(toEnum(x),toEnum(y),toEnum(z)) | z <- [0..] , y <- [0..z^2], x <- [0..z^2], 2*x^3 + 5*y + 2
== z^2 ]


--Aufgabe 6.2

data STree a = BinS (STree a) a (STree a) | LeftS (STree a) a| RightS a (STree a) | LeafS a


instance Show a => Show (STree a) where

        show (LeafS a) = show(a)

        show (LeftS (a) b) = show(b) ++ "(" ++ show(a) ++ "," ++ ")"

        show (RightS (a) b) = show(a) ++ "(" ++ show(b) ++ "," ++ ")"

        show (BinS l a r) = show(a) ++ "(" ++ show(l) ++ "," ++ show(r) ++ ")"
```

```
--Aufgabe 6.3

data Tree a = V a | F a [Tree a] deriving (Show)


--Aufgabe 6.3 a

treeAnd :: Tree Bool -> Bool

treeAnd (V a) = if a == True then True else False

treeAnd (F a ts) = if a == False then False else and $ map (treeAnd) ts


--Aufgabe 6.3 b


treeZip :: Tree a -> Tree b -> Tree (a,b)

treeZip (V a) (V b) = (V (a,b))

treeZip (F a ts) (F b xs) = F (a,b) $ map' (treeZip) ts xs

treeZip (F a ts) (V b) = V (a,b)

treeZip (V a) (F b xs) = V (a,b)


map' f (a:as) (x:xs) = f a x : map' f as xs

map' _ _ _    = []
```