```haskell
type Alias = [Int]          f :: Alias -> [Bool]
type Seznam a = [a]
```

Abstraktní datový typ (ADT)

```haskell
data Colour = Red | Orange | Yellow | Green | Blue | Violet deriving (Show, Eq)
                                                                        | Ord
```

název typu — konstruktory

```haskell
data Bool = True | False
data Int32 = -2^31 | -2^31-1 | ... | 0 | 1 | 2 | ... | 2^31-1
```

```haskell
colourHue :: Colour -> Int
colourHue c |
    c == Red = 0

colourHue Red = 0
colourHue Orange = 30
:

fib 0 = 0
fib 1 = 1
fib x = fib (x-1) + fib (x-2)
```

```haskell
data Name = Name String String deriving (Show, Eq)

Name "Petr" "Novák" :: Name
(a,b)        (,) :: a -> b -> (a,b)

firstName (Name s _) = s
surname (Name _ s) = s
createNames a b = [Name x y | x <- a, y <- b]

data Person = Person Name Int Gender
nameP (Person n _ _) = n
age (Person _ a _) = a
gender (Person _ _ g) = g
-- fst
-- snd

null l = case l of
    [] = True
    (x:xs) = False

my_take _ Empty = Empty
my_take n (Cons x xs)
    | n==0 = Empty
    | otherwise = Cons x (my_take (n-1) xs)

-- [] ==> Empty
-- (x:xs) ==> Cons x xs

data Date = Date {
    day :: Int,
    month :: Int,
    year :: Int } deriving (Ord)

fromStdList = foldr Cons Empty
```

```haskell
class MyEq a where
    eq, neq :: a -> a -> Bool
    neq x y = not $ eq x y

instance MyEq T where
    eq a b = ...

myElem :: (MyEq a) => a -> [a] -> Bool

instance MyOrd Shape where
    lt x y = (obsah x) < (obsah y)
    gt x y = (obsah x) > (obsah y)

sort :: (MyOrd a) => List a -> List a
```

Maybe

```haskell
data Maybe a = Nothing | Just a

data Fraction = Frac Integer Integer

headSafe :: [a] -> Maybe a
headSafe [] = Nothing
headSafe (x:xs) = Just x

maybeZip [] [] = []
maybeZip a b = (headSafe a, headSafe b) : (maybeZip (drop 1 a)
                                                    (drop 1 b))
```