HASKELL-**patterns**

$$
\begin{array}{lcl}
\underline{\text{pat}} & \rightarrow & \underline{\text{var}} \\
& | & \text{\_} \\
& | & \underline{\text{integer}} \\
& | & \underline{\text{float}} \\
& | & \underline{\text{char}} \\
& | & \underline{\text{string}} \\
& | & (\underline{\text{constr}}\ \underline{\text{pat}}_1\ \ldots\ \underline{\text{pat}}_n) \\
& | & [\underline{\text{pat}}_1, \ldots, \underline{\text{pat}}_n], \quad n \geq 0 \\
& | & (\underline{\text{pat}}_1, \ldots, \underline{\text{pat}}_n), \quad n \geq 0
\end{array}
$$

HASKELL-**types**

$$
\begin{aligned}
\underline{\text{type}} \quad \rightarrow \quad & (\underline{\text{tyconstr}}\ \underline{\text{type}}_1 \ldots \underline{\text{type}}_n), \quad n \geq 0 \\
\mid\ & [\underline{\text{type}}] \\
\mid\ & (\underline{\text{type}}_1 \rightarrow \underline{\text{type}}_2) \\
\mid\ & (\underline{\text{type}}_1, \ldots, \underline{\text{type}}_n), \quad n \geq 0 \\
\mid\ & \underline{\text{var}}
\end{aligned}
$$

$$
\underline{\text{tyconstr}} \quad \rightarrow \quad \text{string starting with upper case symbol}
$$

Top declarations and type introduction

$$\underline{\text{topdecl}} \rightarrow \quad \underline{\text{decl}}$$

$$\mid \texttt{type}\ \underline{\text{tyconstr}}\ \underline{\text{var}}_1\ \ldots\ \underline{\text{var}}_n\ =\ \underline{\text{type}}, \quad n \geq 0$$

$$\mid \texttt{data}\ \underline{\text{tyconstr}}\ \underline{\text{var}}_1\ \ldots\ \underline{\text{var}}_n\ =$$

$$\underline{\text{constr}}_1\ \underline{\text{type}}_{1,1} \ldots \underline{\text{type}}_{1,n_1} \mid$$
$$\vdots$$
$$\underline{\text{constr}}_k\ \underline{\text{type}}_{k,1} \ldots \underline{\text{type}}_{k,n_k},$$

$$n \geq 0, k \geq 1, n_i \geq 0$$

Definition of simple data structures

```
data Color = Red | Yellow | Green

data MyBool = MyTrue | MyFalse


traffic_light :: Color -> Color
traffic_light Red    = Green
traffic_light Green  = Yellow
traffic_light Yellow = Red


und :: MyBool -> MyBool -> MyBool
und MyTrue y = y
und _      _ = MyFalse
```

Definition of natural numbers

```
data Nats = Zero | Succ Nats
```

```
plus :: Nats -> Nats -> Nats
plus Zero     y = y
plus (Succ x) y = Succ (plus x y)

half :: Nats -> Nats
half Zero             = Zero
half (Succ Zero)      = Zero
half (Succ (Succ x)) = Succ (half x)
```

Definition of lists

```
data List a = Nil | Cons a (List a)


len :: List a -> Nats
len Nil          = Zero
len (Cons x xs) = Succ (len xs)


append :: List a -> List a -> List a
append Nil          ys = ys
append (Cons x xs) ys = Cons x (append xs ys)
```