Patterns are special expressions that are used to describe the forms of expected arguments.

\[
\begin{align*}
\text{len} &: \text{[Int]} \rightarrow \text{Int} \\
\text{len} \ [\ ] &= 0 \\
\text{len} \ (x : xs) &= 1 + \text{len} \ xs
\end{align*}
\]

\[
\begin{align*}
\text{append} &: \text{[Int]} \rightarrow \text{[Int]} \rightarrow \text{[Int]} \\
\text{append} \ [] \ ys &= ys \\
\text{append} \ (x : xs) \ ys &= x : \text{append} \ xs \ ys
\end{align*}
\]

\[\text{append} \text{ is pre-defined as an infix function called } ++:\]

\[
[1, 2] ++ [3, 4, 5] = [1, 2, 3, 4, 5]
\]

\[
\text{len} \ (\text{append} \ [1] \ [2]) = \quad \text{to evaluate len, we need to evaluate}
\]

\[
\text{append} \text{ a few steps until we know}
\]

\[
\text{whether the first defining equation of len matches}
\]

\[
\begin{align*}
\text{zeros} &: \text{[Int]} \\
\text{zeros} &= 0 : \text{zeros}
\end{align*}
\]

\[
\begin{align*}
\text{f} &: \text{[Int]} \rightarrow \text{[Int]} \rightarrow \text{[Int]} \\
\text{f} \ [\ ] \ ys &= [\ ] \\
\text{f} \ xs \ [\ ] &= [\ ]
\end{align*}
\]

\[
\text{f} \ [\ ] \text{zeros} = [\ ] \text{ Terminates}
\]

\[
\text{f} \ \text{zeros} \ [\ ] = \text{f} \ (0 : \text{zeros}) \ [\ ] = [\ ] \text{ Terminates}
\]

\[\text{at this point, one}
\]

\[\text{notices that the pattern } [\ ]
\]

\[\text{in the first } \text{f-equation does}
\]

\[\text{not match.} \]
Patterns have to be linear, i.e., no variable may occur twice on the lhs of a defining equation.

Reason:

\[
\text{equal :: } [\text{Int}] \rightarrow [\text{Int}] \rightarrow \text{Bool}
\]

\[
\begin{align*}
\text{equal } xs \; & xs \; = \; \text{True} \\
\text{equal } xs \; (x : xs) \; & = \; \text{False}
\end{align*}
\]

Up to now: evaluation strategy influences termination and efficiency, but not the result.

Here: evaluation strategy does influence result:

\[
\begin{align*}
\text{equal } \text{zeros } & \text{zeros} \\
& \quad \land \\
& \quad \text{equal } \text{zeros } (0 : \text{zeros}) \\
& \quad \land \\
& \quad \text{False}
\end{align*}
\]

For every form of pattern (Slide 12), we describe which arguments are matched by this pattern and how the variables of the pattern are instantiated.

- **Var**: a variable matches any expression, matching instantiates the variable by the expression

\[
\text{Square } x = x \times x
\]

- **_**: joker pattern, matches any expression, but no variable is instantiated. "_" may occur multiple times, stands for possibly different values
stands for possibly different values

\[ \text{and True } y = y \] \quad \text{and False } \text{True} = \text{False} \\
\text{and } _{-} \ _{-} = \text{False} \\

- integer, float, char, string: these patterns only match themselves

\[ \text{is}\_5 :: \text{Int} \to \text{Bool} \]
\[ \text{is}\_5 5 = \text{True} \]
\[ \text{is}\_5 _{-} = \text{False} \]

- \( (\text{Constr } \text{pat}_1 \ldots \text{pat}_n) \), \( n \geq 0 \)
  - data constructor (e.g. \text{True}, \text{[], [], ...})
  - where \text{pat}_i\text{ matches } \text{exp}_i

  \[ \text{e.g. } \text{len } (x : xs) = 1 + \text{len } xs \]

- \( [\text{pat}_1, \ldots, \text{pat}_n] \), \( n \geq 0 \) matches all lists

  \[ [\text{exp}_1, \ldots, \text{exp}_n] \text{ where } \text{pat}_i\text{ matches } \text{exp}_i \]

  \[ \text{has}_\text{length}_\text{three} :: [\text{Int}] \to \text{Bool} \]

  \[ \text{has}_\text{length}_\text{three} [_, _, _, _] = \text{True} \]

  \[ \text{has}_\text{length}_\text{three } _{-} = \text{False} \]

- \( (\text{pat}_1, \ldots, \text{pat}_n) \) where \( n \geq 0 \) matches all tuples

  \( (\text{exp}_1, \ldots, \text{exp}_n) \text{ where } \text{pat}_i\text{ matches } \text{exp}_i \).
\[
\text{maxi} : (\text{Int, Int}) \rightarrow \text{Int}
\]
\[
\text{maxi} (0, y) = y
\]
\[
\text{maxi} (x, 0) = x
\]
\[
\text{maxi} (x, y) = x + \text{maxi} (x - 1, y - 1)
\]