From Haskell to Prolog (1)

\[
\begin{align*}
\text{head} &: \; [a] \rightarrow a \\
\text{head} (x:_&) &= x \\
\text{tail} &: \; [a] \rightarrow [a] \\
\text{tail} (_:xs) &= xs \\
\text{null} &: \; [a] \rightarrow \text{Bool} \\
\text{null} [] &= \text{True} \\
\text{null} (_:_&) &= \text{False}
\end{align*}
\]

- head([X|_],X).
- tail([_|Xs],Xs).
- null([]).

```prolog
head([X|_],X).
tail([_|Xs],Xs).
null([]).
```

```haskell
head :: [a] -> a
head (x:__) = x

tail :: [a] -> [a]
tail (_:xs) = xs

null :: [a] -> Bool
null [] = True
null (_:__) = False
```
```
last :: [a] -> a
last [x] = x
last (_:xs) = last xs

init :: [a] -> [a]
init [] = []
init (x:xs) = x : init xs

last([X],X).
last([_|Xs],Y):-last(Xs,Y).

init([_|[]]).
init([X|Xs],[X|Ys]):-
    init(Xs,Ys).
```
From Haskell to Prolog (3)

length :: [a] -> Int
length []  = 0
length (_:l) = 1 + length l

sumList :: (Num a) => [a] -> a
sumList [] = 0
sumList (x:xs) = x + sumList xs

nth :: Int -> [a] -> a
nth 0 (x:_ ) = x
nth n (_:xs) |
  n > 0 = nth (n-1) xs

length([],0).
length([],N):-length(L,N0),
  N is 1+N0.

sumList([],0).
sumList([X|Xs],N):-
  sumList(Xs,N0),N is X+N0.

nth(0,[X|_],X).
nth(N,[_|Xs],Y):-N>0,
  N1 is N-1,nth(N1,Xs,Y).
From Haskell to Prolog (4)

\[
\begin{align*}
take & :: \text{Int} \rightarrow [a] \rightarrow [a] \\
take \ 0 \ _ & = [] \\
take \ _ \ [] & = [] \\
take \ n \ (x:xs) & \\
& | \ n > 0 = x : \text{take} \ (n-1) \ xs \\
\end{align*}
\]

\[
\begin{align*}
drop & :: \text{Int} \rightarrow [a] \rightarrow [a] \\
drop \ 0 \ xs & = xs \\
drop \ _ \ [] & = [] \\
drop \ n \ (_:xs) & \\
& | \ n > 0 = \text{drop} \ (n-1) \ xs \\
\end{align*}
\]

\[
\begin{align*}
\text{take}(0,_,[]). \\
\text{take}(_,[],[]). \\
\text{take}(N,[X|Xs],[X|Ys]):-N>0, \\
\text{N1} \ \text{is} \ N-1,\text{take}(\text{N1},Xs,Ys). \\
\end{align*}
\]

\[
\begin{align*}
\text{drop}(0,Xs,Xs). \\
\text{drop}(_,[],[]). \\
\text{drop}(N,[_|Xs],Ys):-N>0, \\
\text{N1} \ \text{is} \ N-1,\text{drop}(\text{N1},Xs,Ys). \\
\end{align*}
\]
splitAt :: Int->[a]->([[a],[a]])
splitAt 0 xs = ([] ,xs)
splitAt _ [] = ([],[])  
splitAt n (x:xs)
  | n > 0  = (x:xs',xs'') where (xs',xs'') = splitAt (n-1) xs
  | otherwise = splitAt (n,[] ,[],xs)
From Haskell to Prolog (6)

```
member :: (Eq a) => a->[a]->Bool
member x [] = False
member x (y:ys) = x == y ||
  member x ys

append :: [a] -> [a] -> [a]
append [] ys = ys
append (x:xs) ys =
  x : append xs ys
```

```
member(X,[X|_]).
member(X,[_|Ys]):-member(X,Ys).

append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]):-
  append(Xs,Ys,Zs).
```
From Haskell to Prolog (7)

\[\text{nreverse} :: [a] \rightarrow [a]\]
\[\text{nreverse} \ [] = []\]
\[\text{nreverse} \ (x:xs) = \]
\[
\begin{align*}
\text{append} \ (\text{nreverse} \ xs) \ [x]
\end{align*}
\]

\[\text{reverse} :: [a] \rightarrow [a]\]
\[\text{reverse} \ xs = \text{rev} \ xs \ []\]

\[\text{rev} :: [a] \rightarrow [a] \rightarrow [a]\]
\[\text{rev} \ [] \ ys = ys\]
\[\text{rev} \ (x:xs) \ y0s = \text{rev} \ xs \ (x:y0s)\]

\[\text{nreverse}([],[]).\]
\[\text{nreverse}([X|Xs],Ys):-\]
\[\quad \text{nreverse}(Xs,Ys1),\]
\[\quad \text{append}(Ys1,[X],Ys).\]

\[\text{reverse}(Xs,Ys):-\text{rev}(Xs,[],Ys).\]

\[\text{rev}([],Ys,Ys).\]
\[\text{rev}([X|Xs],Y0s,Ys):-\]
\[\quad \text{rev}(Xs,[X|Y0s],Ys).\]
maxList :: (Ord a) => [a] -> a
maxList xs = maxL xs 0

maxL :: (Ord a) => [a] -> a -> a
maxL [] n = n
maxL (x:xs) n0 |
| x > n0 = maxL xs x
| otherwise = maxL xs n0

maxList(Xs,N):=maxL(Xs,0,N).

maxL([],N,N).
maxL([X|Xs],NO,N):-
    ( X>NO -> maxL(Xs,X,N)
    ; otherwise -> maxL(Xs,NO,N)
    ).