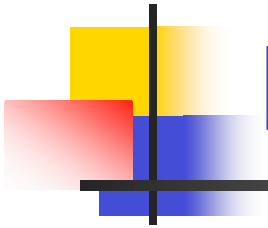


From Haskell to Prolog (1)

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

```
head([X|_],X).
```

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

```
tail([_|Xs],Xs).
```

```
null :: [a] -> Bool
null []     =  True
null (_:_)  =  False
```

```
null([]).
```

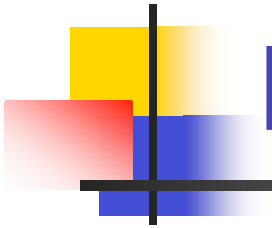
From Haskell to Prolog (2)

```
last :: [a] -> a
last [x]    =  x
last (_:xs) =  last xs
```

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

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

```
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([_|L],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)

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

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

```
take(0,_,[]).
take(_,[],[]).
take(N,[X|Xs],[X|Ys]):-N>0,
   N1 is N-1,take(N1,Xs,Ys).

drop(0,Xs,Xs).
drop(_,[],[]).
drop(N,[_|Xs],Ys):-N>0,
   N1 is N-1,drop(N1,Xs,Ys).
```

From Haskell to Prolog (5)

```
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
```

```
splitAt(0,Xs,[],Xs).
splitAt(_,[],[],[]).
splitAt(N,[X|Xs],[X|Xs1],Xs2):-
   N>0,N1 is N-1,
   splitAt(N1,Xs,Xs1,Xs2).
```

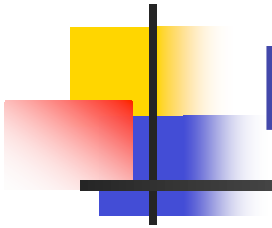
From Haskell to Prolog (6)

```
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)

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


reverse :: [a] -> [a]
reverse xs = rev xs []

rev :: [a] -> [a] -> [a]
rev [] ys       = ys
rev (x:xs) y0s = rev xs (x:y0s)
```

```
nreverse([],[]).
nreverse([X|Xs],Ys):-
   nreverse(Xs,Ys1),
   append(Ys1,[X],Ys).


reverse(Xs,Ys):-rev(Xs,[],Ys).


rev([],Ys,Ys).
rev([X|Xs],YOs,Ys):-
   rev(Xs,[X|YOs],Ys).
```

From Haskell to Prolog (8)

```
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],N0,N):-
      ( X>N0 ->       maxL(Xs,X,N)
      ; otherwise -> maxL(Xs,N0,N)
      ).
```