1. mylengthr [1,2,3] => myfoldr (_ n -> 1+n) 0 [1,2,3] => f 1 (myfoldr f 0 [2,3]
   => f 1 (f 2 (myfoldr f 0 [3])) => f 1 (f 2 (f 3 (0))) where f = (_ n -> 1+n)
   f 1 (f 2 (f 3 (0))) => f 1 (f 2 (1)) => f 1 (2) => 3
2. **myfoldl :: (a -> b -> a) -> a -> [b] -> a**
   myfoldl f acc [] = acc
   myfoldl f acc (x:xs) = myfoldl f (f acc x) xs
   a. mylengthl :: [a] -> Int
      mylengthl = myfoldl 0 (\n -> n - 1) 0
   b. mylengthl [1,2,3] => myfoldl (_ n -> n+1) 0 [1,2,3] => let f = (_ n -> n + 1)
      => myfoldl f (f 0 1) [2, 3] => myfoldl f 1 [2,3] => myfoldl f (f 0 2) [3]
      => myfoldl f 2 [3] => myfoldl f (f 0 3)[] => myfoldl f 3 => 3
3. **myfoldl :: (a -> b -> a) -> a -> [b] -> a**
   myfoldl f acc [] = acc
   myfoldl f acc (x:xs) = myfoldl f (f acc x) xs
   a. myreverse :: [a] -> [a]
      myreverse [] = []
      myreverse (x:xs) = (myreverse xs) ++ ([(myfoldl (_ _ -> 0) x [])])
   b. myreverse [1,2,3] => let f = myfoldl (_ _ -> 0) =>
      (myreverse [2,3]) ++ ([(f 1 [])]) => (myreverse [2,3]) ++ [1] =>
      (myreverse [3]) ++ ([(f 2)]) ++ [1] => ((myreverse [3]) ++ [2]) ++ [1] =>
      ((myreverse []) ++ ([(f 3)]) ++ [2] ++ [1] => ((([]))) ++ [3] ++ [2] ++ [1] =>
      [3,2,1]