```haskell
module Haskell_Mini_Project
( tail'
, sum'
, distance
, sum''
, reverse'
, reverse''
, pack
) where

import           Assist_Lib

-- 1 --
tail' :: [a] -> [a]
tail' [] = []
tail' (_:theList) = theList

-- 2 --
sum' :: (Num a) => [a] -> a
sum' [] = 0
sum' (l:theList) = l + sum' theList

-- 3 --
distance :: (Integral a, Floating b) => (a, a) -> (a, a) -> b
distance (x1, y1) (x2, y2) = sqrt . fromIntegral . sum
      $ map (^2) [(x1 - x2), (y1 - y2)]

-- 4 --
sum'' :: (Num a) => [a] -> [a] -> [a]
sum'' _ [] = []
sum'' [] _ = []
sum'' (f:firstList) (s:secondList) =
      (f + s) : (sum'' firstList secondList)

-- 5 --
reverse’a :: [a] -> [a]
reverse’a [] = []
reverse’a theList = (last theList)
      : (reverse' $ init theList)

reverse’b :: [a] -> [a]
reverse’b [] = []
reverse’b (l:theList) = reverse' theList ++ [l]
```

```haskell
-- 6 --
pack :: [(a, Bool)] -> [(a, Bool)]
pack dataBlocks = [ validData
      | validData <- dataBlocks,
        not $ snd validData]
```

```haskell
module Assist_Lib
( store
, head'
, init'
, last'
, points
) where

import           System.Random

head' :: [a] -> a
head' (theHead:_) = theHead

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

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

deleteMap = randoms (mkStdGen 1) :: [Bool]

store :: [a] -> [(a, Bool)]
store [] = []
store theData = zip theData deleteMap

points = [(4, 3), (1, 6), (2, 4)]
```