```haskell
-- Model solution for Lab05
--
-- Copyright [2000..2004] Manuel M T Chakravarty

module Lab05
where

-- Tests whether a list is a superlist of another one
--
-- Examples: [3,2,1] `superlist` [1..2] = True
--           [3,1]   `superlist` [1..2] = False
--           [3,1]   `superlist` []     = True
--
superlist           :: Eq a => [a] -> [a] -> Bool
superlist _  []      = True
superlist ys (x:xs)  = x `elem` ys && superlist ys xs

-- Split a list into two depending on whether the elements are smaller or
-- bigger than a fixed number
--
-- Examples: split 6 [5,3,6,8,9,3,2,1,4,7,8,9,7] =
--             ([5,3,6,3,2,1,4],[8,9,7,8,9,7])
--           split 9 [1..10]                      = ([1,2,3,4,5,6,7,8,9],[10])
--
split                               :: Int -> [Int] -> ([Int], [Int])
split median []                      = ([], [])
split median (x:xs) | x <= median    = let (ss, gs) = split median xs
                                       in
                                       (x:ss, gs)
                    | otherwise      = let (ss, gs) = split median xs
                                       in
                                       (ss, x:gs)

-- An alternative version
--
-- We didn't discuss this syntax in the lecture, but a `where` clause is
-- valid in all preceding alternatives that are distinguished by guards (this
-- leads to a shorter and more elegant solution to split); moreover, the
-- definition works, in fact, on all types that have an ordering (not only
-- integers)
--
split'                              :: Ord a => a -> [a] -> ([a], [a])
split' median []                     = ([], [])
split' median (x:xs) | x <= median   = (x:ss, gs)
                     | otherwise     = (ss, x:gs)
                       where
                         (ss, gs) = split' median xs

-- Quicksort
--
qsort        :: Ord a => [a] -> [a]
qsort []      = []
qsort (x:xs)  = qsort ss ++ [x] ++ qsort gs
                where
                  (ss, gs) = split' x xs

-- Movie database
--
type MovieList = [(String, Bool)]  -- title, rented

movies = [("The Matrix"  , False),
          ("Strange Days", True ),
          ("Blade Runner", False)]

-- Mark a newly rented movie in the database
--
-- Examples: rent movies "Blade Runner" =
--             [("The Matrix",False),("Strange Days",True),
--              ("Blade Runner",True)]
```

```haskell
--           rent movies "Strange Days"
--             => error message
--           rent movies "Final Fantasy"
--             => error message
--
rent                                    :: MovieList -> String -> MovieList
rent [] movie                            = error "rent: unknown movie"
rent ((movie', rented) : movies) movie
  | movie' == movie && rented            = error "rent: movie already rented"
  | movie' == movie && not rented        = (movie', True)    : movies
  | otherwise                            = (movie', rented) : rent movies movie
```