```
> # The fucntion creates a special matrix object that can cache its inverse
> makeCacheMatrix <- function(x = matrix()) {
+   inv <- NULL
+   set <- function(y) {
+     x <<- y
+     inv <<- NULL
+   }
+   get <- function() x
+   setInverse <- function(inverse) inv <<- inverse
+   getInverse <- function() inv
+   list(set = set,
+        get = get,
+        setInverse = setInverse,
+        getInverse = getInverse)
+ }
> # The function computes the inverse of the special matrix created by makeCa
cheMatrix above. If the inverse has already been calculated, then it should r
etrieve the inverse the the cache.
> cacheSolve <- function(x, ...) {
+   ## Return a matrix that is the inverse of 'x'
+   inv <- x$getInverse()
+   if (!is.null(inv)) {
+     message("getting cached data")
+     return(inv)
+   }
+   mat <- x$get()
+   inv <- solve(mat, ...)
+   x$setInverse(inv)
+   inv
+ }
> # Testing a fuction
> my_matrix <- makeCacheMatrix(matrix(1:4, 2,2))
> my_matrix$get()
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> my_matrix$getInverse()
NULL
> cacheSolve(my_matrix)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> cacheSolve(my_matrix)
getting cached data
     [,1] [,2]
```

```
[1,]   -2  1.5
[2,]    1 -0.5
> my_matrix$getInverse()
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> my_matrix$set(matrix(c(2, 2, 1, 4), 2, 2))
> my_matrix$get()
     [,1] [,2]
[1,]    2    1
[2,]    2    4
> my_matrix$getInverse()
NULL
> cacheSolve(my_matrix)
           [,1]       [,2]
[1,]  0.6666667 -0.1666667
[2,] -0.3333333  0.3333333
> cacheSolve(my_matrix)
getting cached data
           [,1]       [,2]
[1,]  0.6666667 -0.1666667
[2,] -0.3333333  0.3333333
> my_matrix$getInverse()
           [,1]       [,2]
[1,]  0.6666667 -0.1666667
[2,] -0.3333333  0.3333333
```