##

## My function "makeCacheMatrix" creates a special "vector", which is really a list containing a function to:

## 1. set the value of the matrix ... (set)

## 2. get the value of the matrix ... (get)

## 3. set the inverse of the matrix ... (setInv)

## 4. get the inverse of the matrix ... (getInv)

##

## My function "cacheSolve" calculates the Inverse of the special "vector" created with the "makeCacheMatrix"

## function. However, it first checks to see if the Inverse has already been calculated. If so, it gets the

## Inverse from the cache and skips the computation. Otherwise, it calculates the Inverse of the data (the

## matrix) and sets the value of the Inverse in the cache via the setInv function.

##

## makeCacheMatrix ------------------------------------

## This function creates 4 functions and returns them in a list, so that they can be called in the global

## environment, once this function is assigned to a "variable"; for example, let's say:

## > f<-makeCacheMatrix()

## > m1<-matrix(1:9,ncol=3)

## > f$set(m1) # stores the matrix "m1"

## > f$get() # gets the matrix that is stored

## > f$setInv(Invm1) # stores the matrix "Invm1", assuming that it is the inverse of the matrix "m1"

## > f$getInv() # gets the inverse of the matrix that has been stored via the "setInv" function

makeCacheMatrix <- function(x = matrix()) {

m= NULL # Initializing the inverse of the matrix

set<-function(y){ # this function stores the matrix given as argument "y"

x<<-y

m<<-NULL # each time this function is called, the "inverse" (m) is initialized

}

get<-function() x # this function will retrieve the matrix that was stored using the "set" function

setInv<-function(invX) m<<-invX # this function stores the matrix given as argument "invX", assuming that it

# is the inverse of the matrix that was stored with the "set" function

getInv<-function() m # this function will retrieve the "inverse" matrix that was stored using the "setInv" function

list(set=set, get=get, setInv = setInv, getInv = getInv) # a list with the functions is returned

}

## cacheSolve ------------------------------------

## This function checks if the inverse of the matrix is already stored and returns it, otherwise it:

## 1. retrieves the matrix that was stored with the "set" function

## 2. calculates its inverse

## 3. stores the inverse of the matrix using the "setInv" function

## 4. returns the inverse matrix

cacheSolve <- function(x, ...) {

## Return a matrix that is the inverse of 'x'

m<-x$getInv() # retrieves the "inverse" of the matrix

if(!is.null(m)){ # validates if the inverse already exists to return it

message("Getting cached data")

return(m)

}

data<-x$get() # if the inverse isn't stored already,retrieves the matrix that was stored with the "set" function

m<-solve(data) # calculates its inverse

x$setInv(m) # stores the inverse of the matrix using the "setInv" function

m # returns the inverse matrix

}