## makeCacheMatrix: This function creates a special "matrix"

## object that can cache its inverse.

makeCacheMatrix<-function(x = matrix())

{

inv<-NULL

set<-function(y)

{

x<<-y

inv<<-NULL

}

## default value

get<-function () x

## setting the Inverse of Object

setinverse<-function(inverse) inv <<- inverse

## Getting Inverse

## Getting My Matrix Object

getinverse<-function() inv

## subsetting Closure of the Object

list(set= set, get = get,

setinverse = setinverse,

getinverse = getinverse)

}

## cacheSolve: This function computes the inverse of the

## special "matrix" returned by makeCacheMatrix above.

## If the inverse has already been calculated (and the matrix

## has not changed), then the cachesolve should retrieve the

## inverse from the cache.

cacheSolve<-function(x, ...)

{

## Getting Inverse of the Object

inv<- x$getinverse()

## testing or coercing the argument for NULL matrix

## is.null () is primitive and is FALSE by default

## hence checking for !is.null()

if (!is.null(inv))

{

return(inv)

}

## Vector Closure

data<-x$get()

## Solve inverses the Object

inv<-solve(data, ...)

x$setinverse(inv)

## returning the value to cacheSolve

inv

}