# Programming Assigment 2

For the second assignment, we need to create 'cache'-functions, so it would be easier and faster to work with large objects.

In a sense, we will redefine the vector and matrix object and the calculation of their mean (for vector) and invetse (for matrix).

*The following code is given:*

```
## makeVector function sets and gets the values of the vector itself and its mean
makeVector <- function(x = numeric()) {
  ## x is the vector object

  m <- NULL # mean
  set <- function(y) {
    x <<- y
    m <<- NULL
  }
  get <- function() x
  setmean <- function(mean) m <<- mean
  getmean <- function() m
  list(set = set, get = get, # the returned values
       setmean = setmean,
       getmean = getmean)
}
```

```
## cachemean function checks is 'm' (mean) is empty, and if so, calculates the mean of the vector and s
cachemean <- function(x, ...) {
        m <- x$getmean()
        if(!is.null(m)) {
                message("getting cached data")
                return(m)
        } # checking if m is null and returning the value
        data <- x$get()
        m <- mean(data, ...)
        x$setmean(m) # else calculating and saving the value
        m
}
```

## Task

The assignment is to create a makeCacheMatrix and cacheSolve functions. For this assignment, we assume that the matrix supplied is always invertible.

By analogy with 'vector'-function we'll make an matrix object named 'i' to store inverse result

```r
## makeVector function sets and gets the values of the matrix itself and its inverse
makeCacheMatrix <- function(x = matrix()) {

  i <- NULL # first, we set inverse to NULL
  set <- function(y){ # set function is identical to vector's one
    x <<- y
    i <<- NULL
  }
  get <- function() x # also similar -- just returning the value
  setinverse <- function(solve) i <<- solve # setting the inverse
  getinverse <- function() i
  list(set = set, get = get, # the returned values
       setinverse = setinverse,
       getinverse = getinverse)
}
```

Now, we create a cache-Solve function, so we could check is the inverse matrix was calculated and saved previously. If it was not, then we calculate the value and place it into 'i' object

```r
## cacheSolve returns a matrix that is the inverse of 'x'
cacheSolve <- function(x) {
  i <- x$getinverse() # calling the get function, created previously
  if(!is.null(i)) {
                message("getting cached data")
                return(i)
        } # checking if i is null and returning the value

  # if cache is empty
  data <- x$get() # getting the data
  i <- solve(data) # calculating the inverse
  x$setinverse(i) # saving inverse in a 'cache' for x object
  i

}
```

A little bit of testing:

```r
v <- makeVector(c(1,2,3,4))
v2 <- makeVector()
v$get()
```

```
## [1] 1 2 3 4
```

```r
v2$get() # v2 should be empty
```

```
## numeric(0)
```

```r
v2$set(c(6,7,8,9))
v2$get() # now it's not empty anymore
```

```
## [1] 6 7 8 9
```

```
v$getmean()
```

```
## NULL
```

```
v2$getmean() # none of the vectors have calculated mean
```

```
## NULL
```

```
v$setmean(2.5) # we can set mean manually
v$getmean()
```

```
## [1] 2.5
```

```
cachemean(v2) # or, by using the cachemean function
```

```
## [1] 7.5
```

Let's look at the larger vectors

```
largev <- makeVector(1:100000)
# we'll measure what time it takes to initially set up the mean value and to get it from cache
start_time <- Sys.time()
cachemean(largev)
```

```
## [1] 50000.5
```

```
after_cache_time <- Sys.time()
cachemean(largev)
```

```
## getting cached data
```

```
## [1] 50000.5
```

```
after_getting_cache_time <- Sys.time()
print(after_cache_time-start_time)
```

```
## Time difference of 0.01176 secs
```

```
print(after_getting_cache_time-after_cache_time)
```

```
## Time difference of 0.003051043 secs
```

It's three times longer to initially calculate the mean of largev than to get it from cache!
Now, vectors testing:

```
m <- makeCacheMatrix(diag(3))
m2 <- makeCacheMatrix()
m2$set(matrix(5:8,nrow=2))
m$get()
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
m2$get()
```

```
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

```
m$getinverse()
```

```
## NULL
```

```
m2$getinverse() # none of the matrices have calculated inverse
```

```
## NULL
```

```
cacheSolve(m)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
cacheSolve(m2) # we'll calculate them using our function
```

```
##      [,1] [,2]
## [1,]   -4  3.5
## [2,]    3 -2.5
```

Now, to larger matrices (results are supressed by invisible() function):

```
largem <- makeCacheMatrix(matrix(1:10000,nrow=100)+diag(100))
largem$getinverse()
```

```
## NULL
```

```
start_time <- Sys.time()
invisible(cacheSolve(largem))
after_cache_time <- Sys.time()
invisible(cacheSolve(largem))
```

```
## getting cached data
```

```r
after_getting_cache_time <- Sys.time()
print(after_cache_time-start_time)
```

```
## Time difference of 0.003913164 secs
```

```r
print(after_getting_cache_time-after_cache_time)
```

```
## Time difference of 0.002730846 secs
```

We can see that it's still a little faster to get value from the cache than to calculate it initially