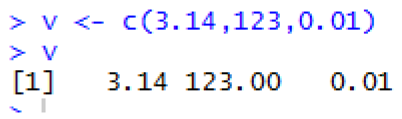
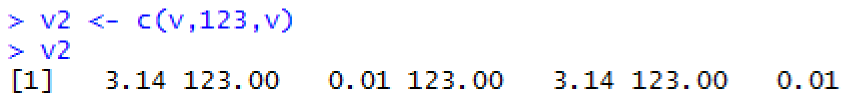
1-Create a vector called v and assign the values (3.14, 123, 0.01)



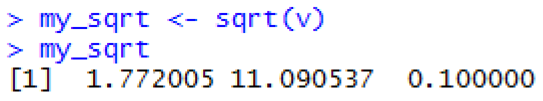
```
> v <- c(3.14,123,0.01)
> v
[1]   3.14 123.00   0.01
```

2- Create a new vector that stores three values: v, 123 and v



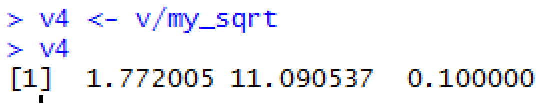
```
> v2 <- c(v,123,v)
> v2
[1]   3.14 123.00   0.01 123.00   3.14 123.00   0.01
```

3- Calculate the square root of vector v and store it in a new variable



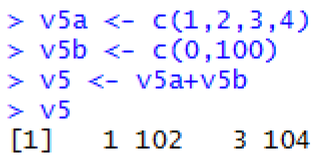
```
> my_sqrt <- sqrt(v)
> my_sqrt
[1]  1.772005 11.090537  0.100000
```

4- Now calculate the value of v divided by my_sqrt



```
> v4 <- v/my_sqrt
> v4
[1]  1.772005 11.090537  0.100000
```

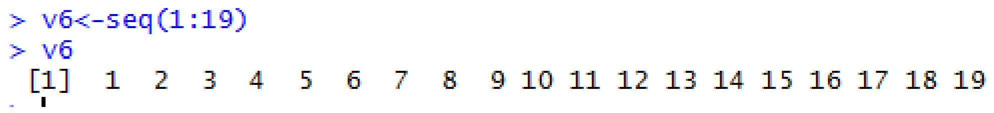
5- Add this two vectors: (1, 2, 3, 4) and (0,100). What do you see? Can two vectors of different length be added?



```
> v5a <- c(1,2,3,4)
> v5b <- c(0,100)
> v5 <- v5a+v5b
> v5
[1]   1 102   3 104
```

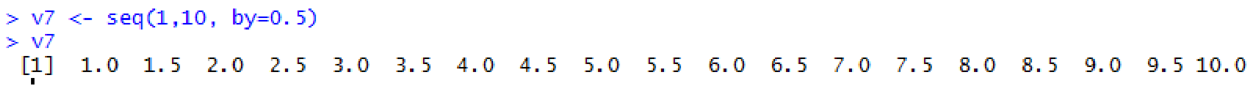
R can sum two vectors with different length, it will repeat the shortest as many times it need

6- Create a sequence of numbers from 1 to 19 using the : operator



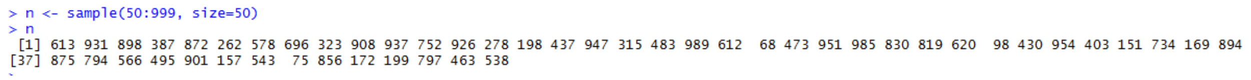
```
> v6<-seq(1:19)
> v6
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

7- Now use seq() to create a sequence from 1 to 10 incrementing by 0.5



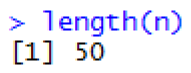
```
> v7 <- seq(1,10, by=0.5)
> v7
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0  9.5 10.0
```

8- Use seq() to generate a list of 50 numbers between 50 and 999 and store the list in a variable n



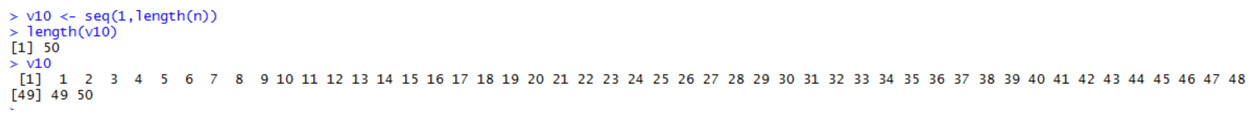
```
> n <- sample(50:999, size=50)
> n
 [1] 613 931 898 387 872 262 578 696 323 908 937 752 926 278 198 437 947 315 483 989 612  68 473 951 985 830 819 620  98 430 954 403 151 734 169 894
[37] 875 794 566 495 901 157 543  75 856 172 199 797 463 538
```

9- Apply the length() function to the variable n. Which is the result?



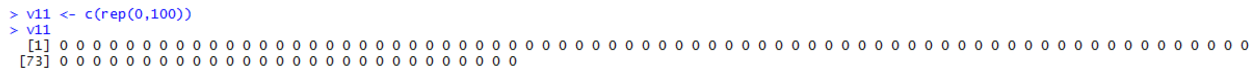
```
> length(n)
[1] 50
```

10- Now generate a new list of numbers starting from 1 that has the same length as n variable



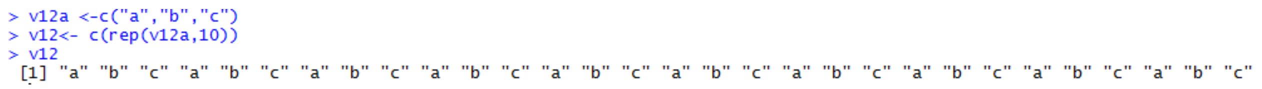
```
> v10 <- seq(1,length(n))
> length(v10)
[1] 50
> v10
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
[49] 49 50
```

11- Create a vector with 100 zeros



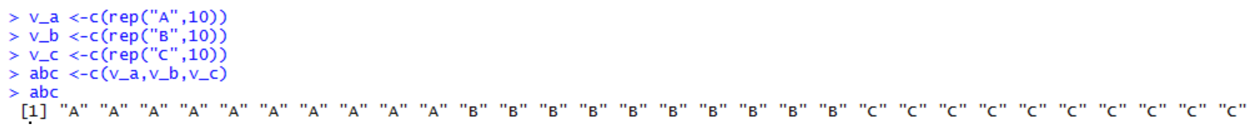
```
> v11 <- c(rep(0,100))
> v11
  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [73] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

12- Create a new vector with 10 repetitions of the vector ("a", "b", "c")



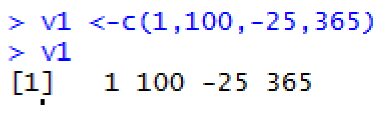
```
> v12a <-c("a","b","c")
> v12<- c(rep(v12a,10))
> v12
 [1] "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

13- Now create a new vector that contains 10 a letters, then 10 b's, then 10 c's and save it into a variable called abc



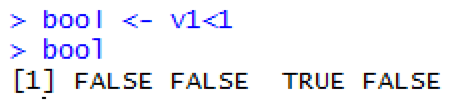
```
> v_a <-c(rep("A",10))
> v_b <-c(rep("B",10))
> v_c <-c(rep("C",10))
> abc <-c(v_a,v_b,v_c)
> abc
 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C"
```

14- Create a vector with the values (1,100,-25,365) and save it into v1 variable



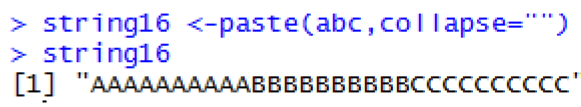
```
> v1 <-c(1,100,-25,365)
> v1
[1]   1 100 -25 365
```

15- Now create a new variable bool that gets the result of v1 "abc"



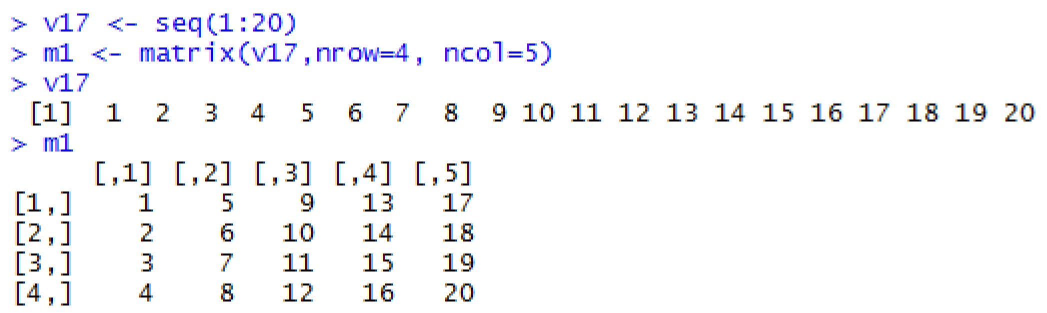
```
> bool <- v1<1
> bool
[1] FALSE FALSE  TRUE FALSE
```

16- Now take abc variable from question 13 and generate a single string value. That is, something like: (a,b,c) -> "abc"



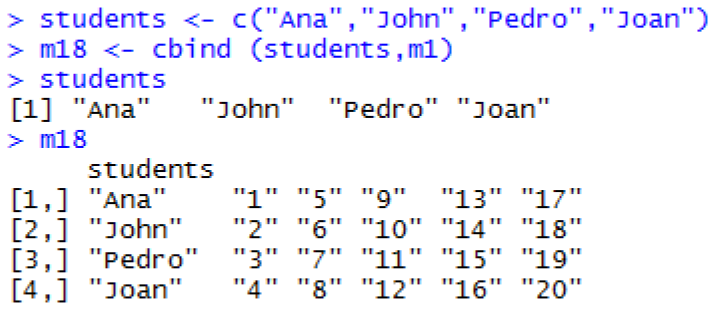
```
> string16 <-paste(abc,collapse="")
> string16
[1] "AAAAAAAAAABBBBBBBBBBCCCCCCCCCC"
```

17- Create a new matrix with 4 rows and 5 columns from a new vector that contains the values from 1 to 20. Store the results in variable m1



```
> v17 <- seq(1:20)
> m1 <- matrix(v17,nrow=4, ncol=5)
> v17
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> m1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

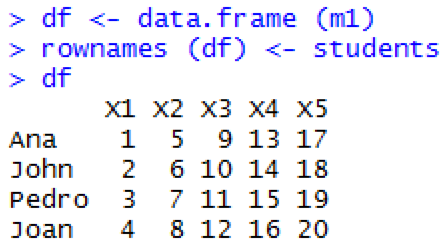
18- Now create a vector called students with the values ("Ana","John","Pedro","Joan") and create a new matrix adding students as a new column to m1 matrix. What happens to numeric values?



```
> students <- c("Ana","John","Pedro","Joan")
> m18 <- cbind (students,m1)
> students
[1] "Ana"   "John"  "Pedro" "Joan" 
> m18
     students                        
[1,] "Ana"    "1" "5" "9"  "13" "17"
[2,] "John"   "2" "6" "10" "14" "18"
[3,] "Pedro"  "3" "7" "11" "15" "19"
[4,] "Joan"   "4" "8" "12" "16" "20"
```

The numeric values has changed to String value, because Matrix can only contain one type of data

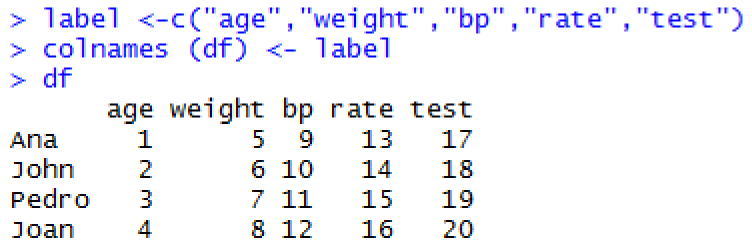
19- Now create a new data.frame() called df with students vector and m1 matrix. What difference do you see with the previous matrix?



```
> df <- data.frame (m1)
> rownames (df) <- students
> df
      X1 X2 X3 X4 X5
Ana    1  5  9 13 17
John   2  6 10 14 18
Pedro  3  7 11 15 19
Joan   4  8 12 16 20
```

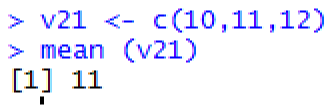
Now we have 2 types of values, integers for the numbers of the matrix and string for the name of the students

20- create a new vector with the labels "name", "age", "weight", "bp", "rate", "test". Now use colnames() to set the colnames attribute of our data frame df. Show your final data frame



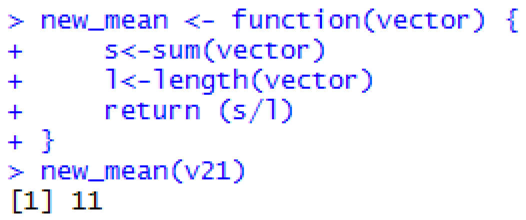
```
> label <-c("age","weight","bp","rate","test")
> colnames (df) <- label
> df
      age weight bp rate test
Ana     1      5  9   13   17
John    2      6 10   14   18
Pedro   3      7 11   15   19
Joan    4      8 12   16   20
```

21- Calculate the mean of a vector with values (10,11,12)



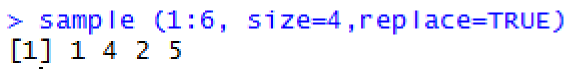
```
> v21 <- c(10,11,12)
> mean (v21)
[1] 11
```

22- Now create a new function call new_mean that receives a vector as a parameter, then calculates the sum of the vector, the length of the vector and return the mean of the vector.



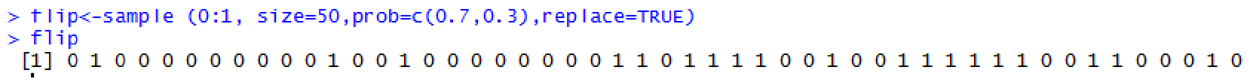
```
> new_mean <- function(vector) {
+     s<-sum(vector)
+     l<-length(vector)
+     return (s/l)
+ }
> new_mean(v21)
[1] 11
```

23- Use sample() function to simulate a rolling dice. That is, randomly select four numbers between 1 and 6 with replacement.



```
> sample (1:6, size=4,replace=TRUE)
[1] 1 4 2 5
```

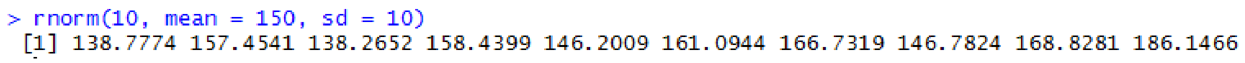
24- Simulate 50 flips of an unfair two-sided coin. The probabilities here are 0.7 for heads and 0.3 for tails. Use sample() to draw a sample of size 50. Store the results in variable flip. How would you test if the results follow the probability defined?



```
> flip<-sample (0:1, size=50,prob=c(0.7,0.3),replace=TRUE)
> flip
 [1] 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 0 1 1 1 1 1 0 0 1 1 0 0 0 1 0
```

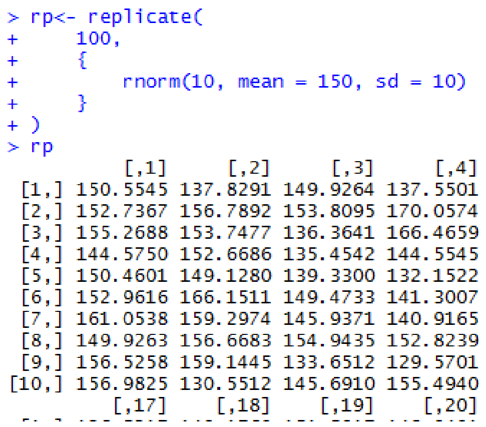
If we want to test if the probability is correct, we need to increment the number of repetitions.

25- How would you generate 10 random numbers from a standard normal distribution with a mean of 150 and a standard deviation of 10?



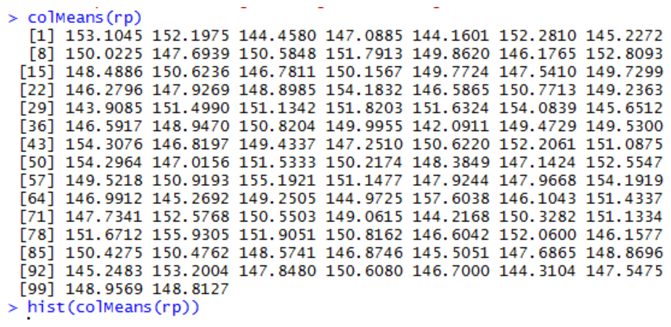
```
> rnorm(10, mean = 150, sd = 10)
 [1] 138.7774 157.4541 138.2652 158.4399 146.2009 161.0944 166.7319 146.7824 168.8281 186.1466
```

26- Now, how would you generate 100 sets of random numbers like the previous question? Save results in a variable called rp

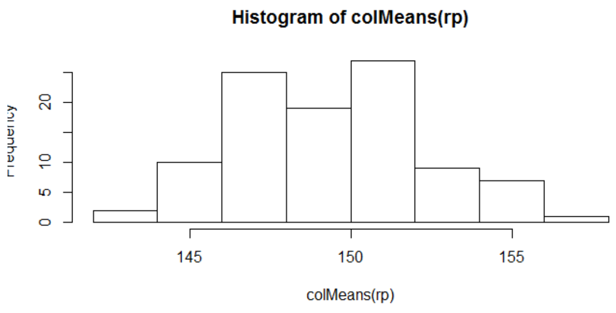


```
> rp<- replicate(
+     100,
+     {
+         rnorm(10, mean = 150, sd = 10)
+     }
+ )
> rp
          [,1]     [,2]     [,3]     [,4]
 [1,] 150.5545 137.8291 149.9264 137.5501
 [2,] 152.7367 156.7892 153.8095 170.0574
 [3,] 155.2688 153.7477 136.3641 166.4659
 [4,] 144.5750 152.6686 135.4542 144.5545
 [5,] 150.4601 149.1280 139.3300 132.1522
 [6,] 152.9616 166.1511 149.4733 141.3007
 [7,] 161.0538 159.2974 145.9371 140.9165
 [8,] 149.9263 156.6683 154.9435 152.8239
 [9,] 156.5258 159.1445 133.6512 129.5701
[10,] 156.9825 130.5512 145.6910 155.4940
         [,17]    [,18]    [,19]    [,20]
```

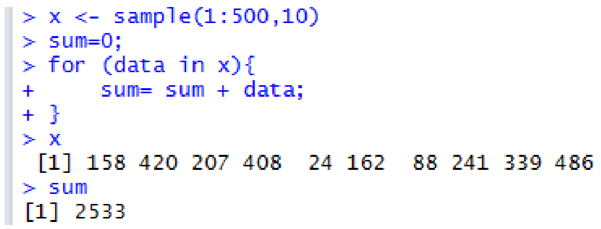
27- Use colMeans() function to find the mean of each column of rp and plot a histogram



```
> colMeans(rp)
  [1] 153.1045 152.1975 144.4580 147.0885 144.1601 152.2810 145.2272
  [8] 150.0225 147.6939 150.5848 151.7913 149.8620 146.1765 152.8093
 [15] 148.4886 150.6236 146.7811 150.1567 149.7724 147.5410 149.7299
 [22] 146.2796 147.9269 148.8985 154.1832 146.5865 150.7713 149.2363
 [29] 143.9085 151.4990 151.1342 151.8203 151.6324 154.0839 145.6512
 [36] 146.5917 148.9470 150.8204 149.9955 142.0911 149.4729 149.5300
 [43] 154.3076 146.8197 149.4337 147.2510 150.6220 152.2061 151.0875
 [50] 154.2964 147.0156 151.5333 150.2174 148.3849 147.1424 152.5547
 [57] 149.5218 150.9193 155.1921 151.1477 147.9244 147.9668 154.1919
 [64] 146.9912 145.2692 149.2505 144.9725 157.6038 146.1043 151.4337
 [71] 147.7341 152.5768 150.5503 149.0615 144.2168 150.3282 151.1334
 [78] 151.6712 155.9305 151.9051 150.8162 146.6042 152.0600 146.1577
 [85] 150.4275 150.4762 148.5741 146.8746 145.5051 147.6865 148.8696
 [92] 145.2483 153.2004 147.8480 150.6080 146.7000 144.3104 147.5475
 [99] 148.9569 148.8127
> hist(colMeans(rp))
```

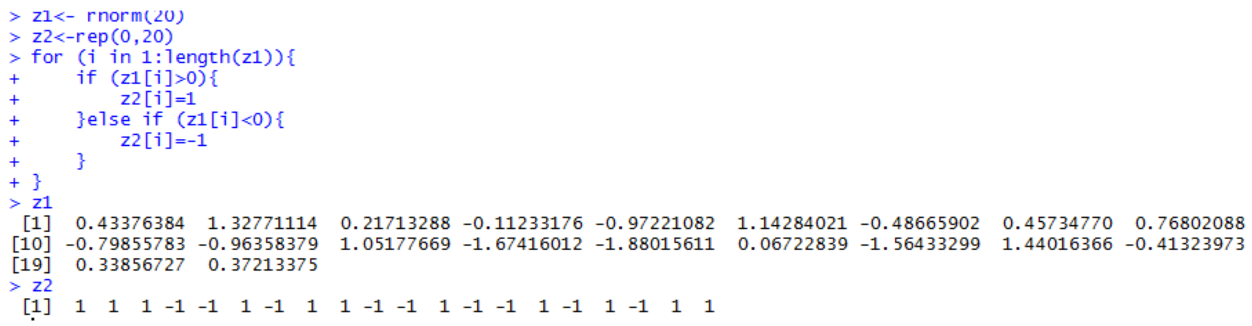


28- Create a vector X with 10 values and calculate the cumulative sums for this vector using a for loop.



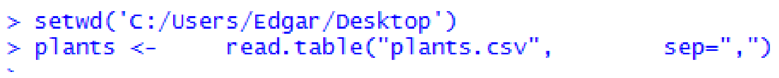
```
> x <- sample(1:500,10)
> sum=0;
> for (data in x){
+      sum= sum + data;
+ }
> x
 [1] 158 420 207 408  24 162  88 241 339 486
> sum
[1] 2533
```

29- Create two vectors, Z1 and Z2. Z1 is a vector of 20 elements filled with 20 random numbers from a standard normal distribution. Z2 is a string of 20 zeros. Design a for loop that counts from 1 to 20. For each value, it has to evaluate if the value is greater than 0. For all those values, it must write a 1 in the same position of the vector Z2. For lower than 0 values it must write a -1



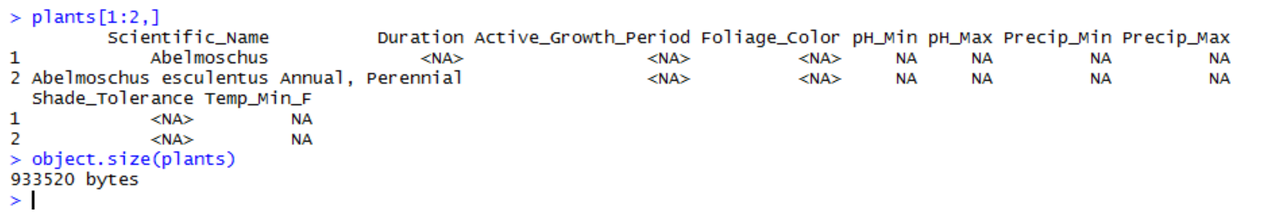
```
> z1<- rnorm(20)
> z2<-rep(0,20)
> for (i in 1:length(z1)){
+     if (z1[i]>0){
+         z2[i]=1
+     }else if (z1[i]<0){
+         z2[i]=-1
+     }
+ }
> z1
 [1]  0.43376384  1.32771114  0.21713288 -0.11233176 -0.97221082  1.14284021 -0.48665902  0.45734770  0.76802088
[10] -0.79855783 -0.96358379  1.05177669 -1.67416012 -1.88015611  0.06722839 -1.56433299  1.44016366 -0.41323973
[19]  0.33856727  0.37213375
> z2
 [1]  1  1  1 -1 -1  1 -1  1  1 -1 -1  1 -1 -1  1 -1  1 -1  1  1
```

30- Now we are going to import a data file:plants.csv and use read.table() function to store the new table in an object called plants



```
> setwd('C:/Users/Edgar/Desktop')
> plants <-     read.table("plants.csv",        sep=",")
```

31- How do you find the rows and columns of the variable plants? And the size in bytes?

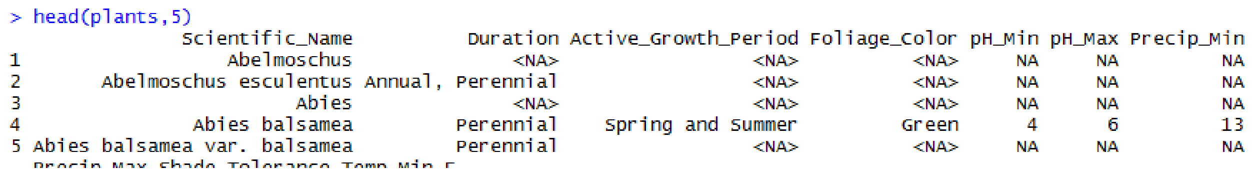


```
> plants[1:2,]
        Scientific_Name          Duration Active_Growth_Period Foliage_Color pH_Min pH_Max Precip_Min Precip_Max
1            Abelmoschus              <NA>                 <NA>          <NA>     NA     NA         NA         NA
2 Abelmoschus esculentus Annual, Perennial                 <NA>          <NA>     NA     NA         NA         NA
  Shade_Tolerance Temp_Min_F
1            <NA>         NA
2            <NA>         NA
> object.size(plants)
933520 bytes
> |
```

We use [row,col] notation. Using the ':' if we want to specify a range. In the example we can see the first 2 rows of the table plants.

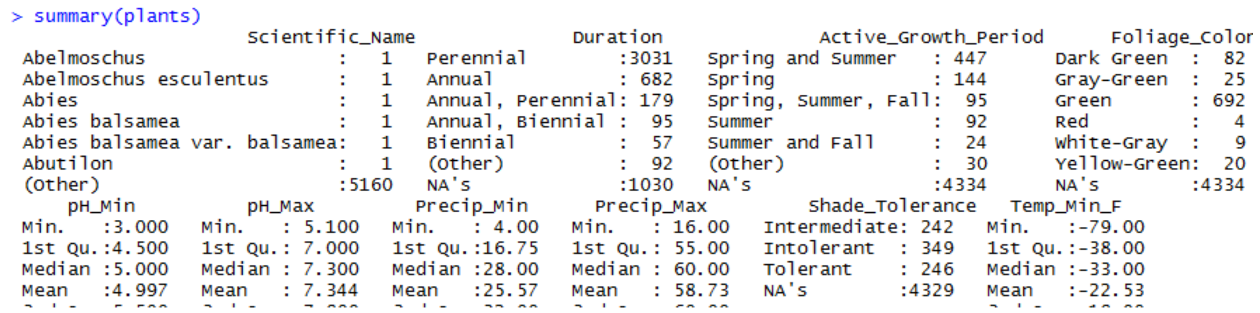
We can know the size using the 'object.size()' command

32- How would you see the first rows of the experiment?



```
> head(plants,5)
             Scientific_Name          Duration Active_Growth_Period Foliage_Color pH_Min pH_Max Precip_Min
1                Abelmoschus              <NA>                 <NA>          <NA>     NA     NA         NA
2     Abelmoschus esculentus Annual, Perennial                 <NA>          <NA>     NA     NA         NA
3                      Abies              <NA>                 <NA>          <NA>     NA     NA         NA
4             Abies balsamea         Perennial    Spring and Summer         Green      4      6         13
5 Abies balsamea var. balsamea         Perennial                 <NA>          <NA>     NA     NA         NA
```

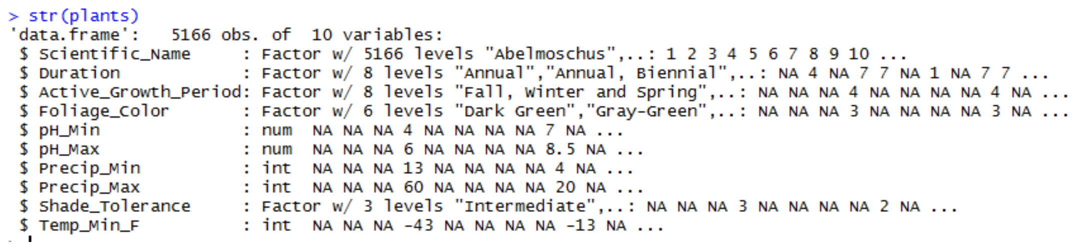
33- What information is summary() providing when applied to the experiment variable? What NA means? What information is str() providing?



```
> summary(plants)
                     Scientific_Name              Duration              Active_Growth_Period      Foliage_Color
 Abelmoschus                 :   1   Perennial         :3031   Spring and Summer   : 447   Dark Green  :  82
 Abelmoschus esculentus      :   1   Annual            : 682   Spring              : 144   Gray-Green  :  25
 Abies                       :   1   Annual, Perennial: 179   Spring, Summer, Fall:  95   Green       : 692
 Abies balsamea              :   1   Annual, Biennial :  95   Summer              :  92   Red         :   4
 Abies balsamea var. balsamea:   1   Biennial         :  57   Summer and Fall     :  24   White-Gray  :   9
 Abutilon                    :   1   (Other)          :  92   (Other)             :  30   Yellow-Green:  20
 (Other)                     :5160   NA's             :1030   NA's                :4334   NA's        :4334
     pH_Min          pH_Max         Precip_Min      Precip_Max       Shade_Tolerance   Temp_Min_F
 Min.   :3.000   Min.   : 5.100   Min.   : 4.00   Min.   : 16.00   Intermediate: 242   Min.   :-79.00
 1st Qu.:4.500   1st Qu.: 7.000   1st Qu.:16.75   1st Qu.: 55.00   Intolerant  : 349   1st Qu.:-38.00
 Median :5.000   Median : 7.300   Median :28.00   Median : 60.00   Tolerant    : 246   Median :-33.00
 Mean   :4.997   Mean   : 7.344   Mean   :25.57   Mean   : 58.73   NA's        :4329   Mean   :-22.53
```

With 'summary()' command we can have a quick view of the table, and the min value, the first qu, median, mean, 3rd qua, max and NA's

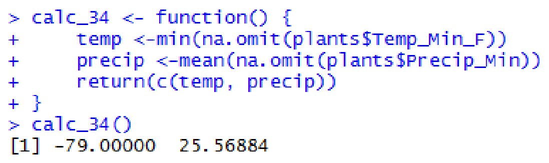
NA means that we don't know the value



```
> str(plants)
'data.frame':	5166 obs. of  10 variables:
 $ Scientific_Name     : Factor w/ 5166 levels "Abelmoschus",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Duration            : Factor w/ 8 levels "Annual","Annual, Biennial",..: NA 4 NA 7 7 NA 1 NA 7 7 ...
 $ Active_Growth_Period: Factor w/ 8 levels "Fall, Winter and Spring",..: NA NA NA 4 NA NA NA NA 4 NA ...
 $ Foliage_Color       : Factor w/ 6 levels "Dark Green","Gray-Green",..: NA NA NA 3 NA NA NA NA 3 NA ...
 $ pH_Min              : num  NA NA NA 4 NA NA NA NA 7 NA ...
 $ pH_Max              : num  NA NA NA 6 NA NA NA NA 8.5 NA ...
 $ Precip_Min          : int  NA NA NA 13 NA NA NA NA 4 NA ...
 $ Precip_Max          : int  NA NA NA 60 NA NA NA NA 20 NA ...
 $ Shade_Tolerance     : Factor w/ 3 levels "Intermediate",..: NA NA NA 3 NA NA NA NA 2 NA ...
 $ Temp_Min_F          : int  NA NA NA -43 NA NA NA NA -13 NA ...
```

Str() give us a view of the structure of the table. What kind of data we have in the object.

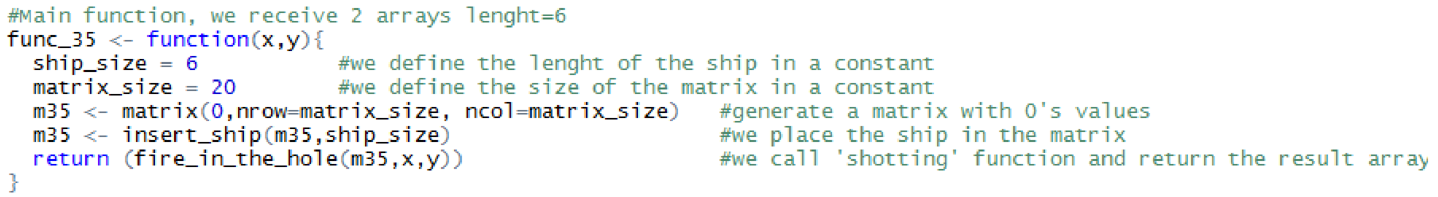
34- Create a function to calculate the minimum Temp_Min_F and the average Precip_Min of the table plants



```
> calc_34 <- function() {
+     temp <-min(na.omit(plants$Temp_Min_F))
+     precip <-mean(na.omit(plants$Precip_Min))
+     return(c(temp, precip))
+ }
> calc_34()
[1] -79.00000  25.56884
```

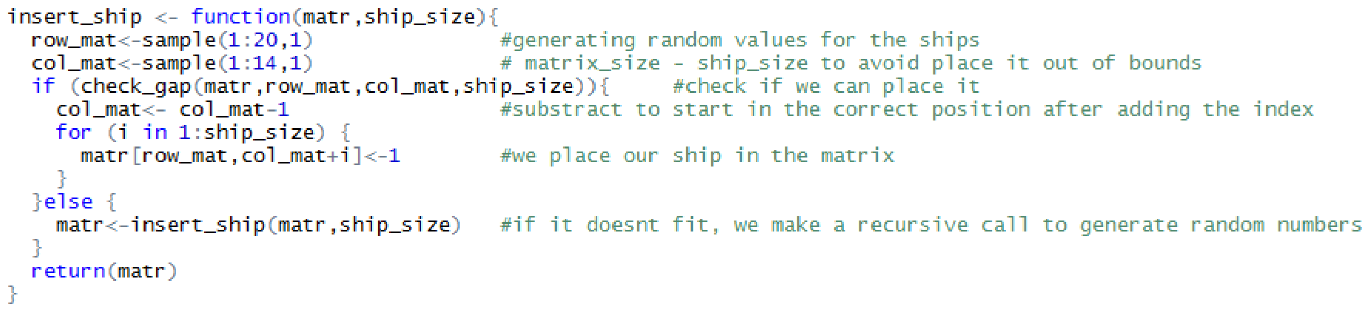
35- Create a function that:

- Creates a 20 by 20 matrix of 0 values
- Places in a random position inside the matrix one vector of size 6 and represents it with six consecutive "1" values. This vector will be our ship
- Receives two vectors of six values as inputs. X and Y. Both vectors store six values between 1 and 20 and represent coordinates in the matrix
- The function returns a vector with six values depending if the coordinates have hit or miss our ship. Example=("hit", "miss", "miss", "miss", "miss", "hit")



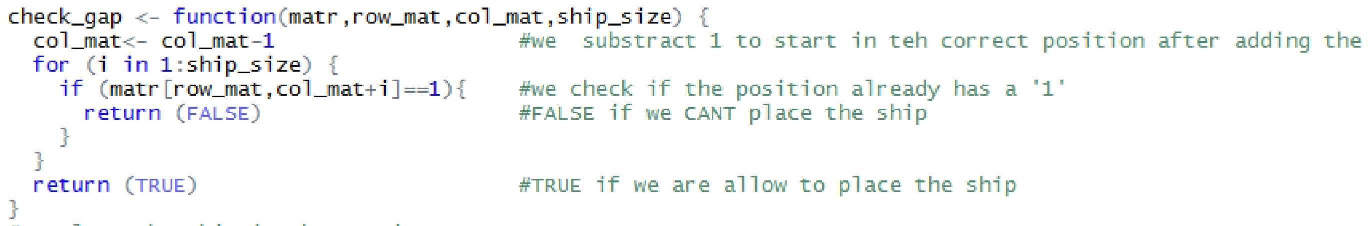
```
#Main function, we receive 2 arrays lenght=6
func_35 <- function(x,y){
  ship_size = 6           #we define the lenght of the ship in a constant
  matrix_size = 20        #we define the size of the matrix in a constant
  m35 <- matrix(0,nrow=matrix_size, ncol=matrix_size)   #generate a matrix with 0's values
  m35 <- insert_ship(m35,ship_size)                     #we place the ship in the matrix
  return (fire_in_the_hole(m35,x,y))                    #we call 'shotting' function and return the result array
}
```

The main function will define the size of the ships, the size of the matrix and it will create the matrix, call a function to put the ships and check the shoots and return the result



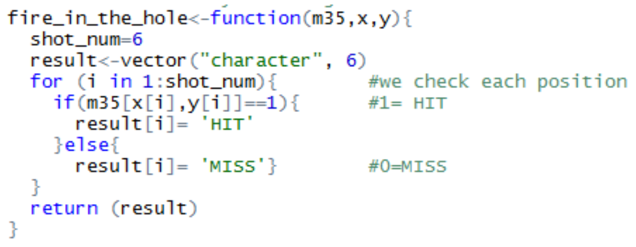
```
insert_ship <- function(matr,ship_size){
  row_mat<-sample(1:20,1)                #generating random values for the ships
  col_mat<-sample(1:14,1)                # matrix_size - ship_size to avoid place it out of bounds
  if (check_gap(matr,row_mat,col_mat,ship_size)){      #check if we can place it
    col_mat<- col_mat-1                  #substract to start in the correct position after adding the index
    for (i in 1:ship_size) {
      matr[row_mat,col_mat+i]<-1         #we place our ship in the matrix
    }
  }else {
    matr<-insert_ship(matr,ship_size)    #if it doesnt fit, we make a recursive call to generate random numbers
  }
  return(matr)
}
```

This function will create the random number for the column and row indexes. Then it will call a function to check if the ship can be placed in the random index or it need to generate random numbers again.



```
check_gap <- function(matr,row_mat,col_mat,ship_size) {
  col_mat<- col_mat-1                  #we  substract 1 to start in teh correct position after adding the
  for (i in 1:ship_size) {
    if (matr[row_mat,col_mat+i]==1){     #we check if the position already has a '1'
      return (FALSE)                     #FALSE if we CANT place the ship
    }
  }
  return (TRUE)                        #TRUE if we are allow to place the ship
}
```

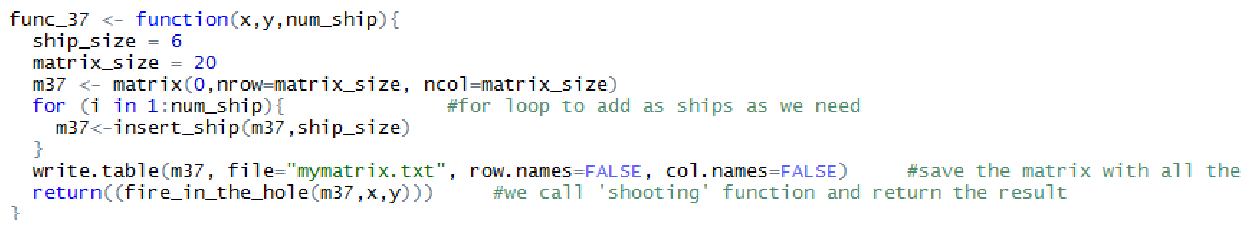
With this function we will check if the ship can be placed in the matrix or not.



```
fire_in_the_hole<-function(m35,x,y){
  shot_num=6
  result<-vector("character", 6)
  for (i in 1:shot_num){          #we check each position
    if(m35[x[i],y[i]]==1){        #1= HIT
      result[i]= 'HIT'
    }else{
      result[i]= 'MISS'}          #0=MISS
  }
  return (result)
}
```

This is the last function, where we check if the shoots array hit any ship or not

37- Make the number of ships in the matrix a parameter given by the user and export the matrix to a text file.



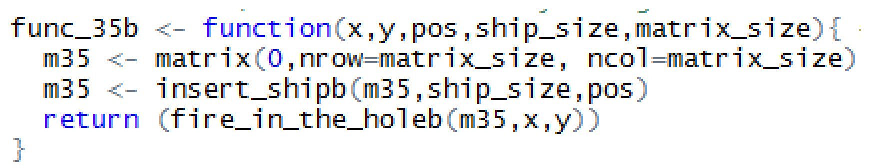
```
func_37 <- function(x,y,num_ship){
  ship_size = 6
  matrix_size = 20
  m37 <- matrix(0,nrow=matrix_size, ncol=matrix_size)
  for (i in 1:num_ship){              #for loop to add as ships as we need
    m37<-insert_ship(m37,ship_size)
  }
  write.table(m37, file="mymatrix.txt", row.names=FALSE, col.names=FALSE)      #save the matrix with all the
  return((fire_in_the_hole(m37,x,y)))      #we call 'shooting' function and return the result
}
```

In this case, we will receive the numbers of ships we want to introduce in the matrix. And then use a loop to insert each ship in the matrix

Bonus

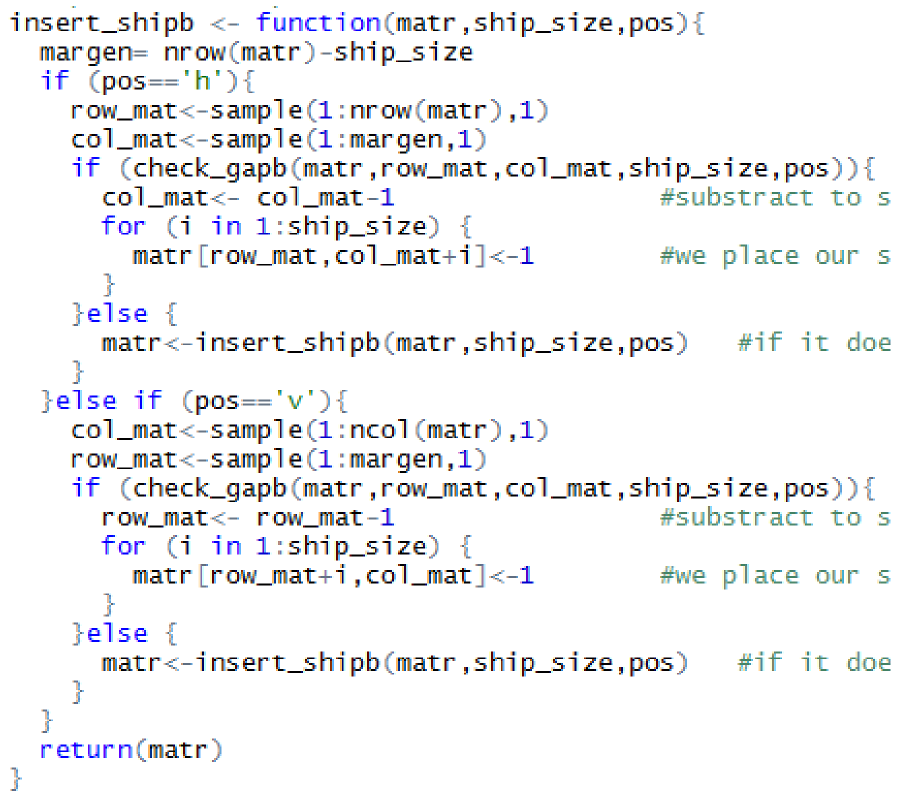
I perform some changes to be able to place the ships in vertical or horizontal, and we avoid the "default" values (matrix size and ship size)

35-



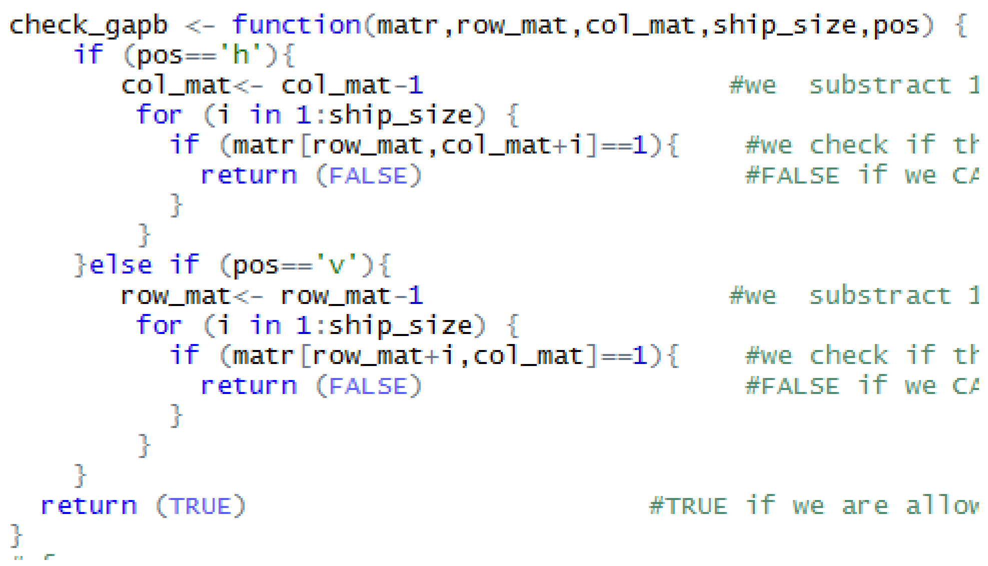
```
func_35b <- function(x,y,pos,ship_size,matrix_size){
  m35 <- matrix(0,nrow=matrix_size, ncol=matrix_size)
  m35 <- insert_shipb(m35,ship_size,pos)
  return (fire_in_the_holeb(m35,x,y))
}
```

We will create the matrix and the ship size concerning the parameters of the function.

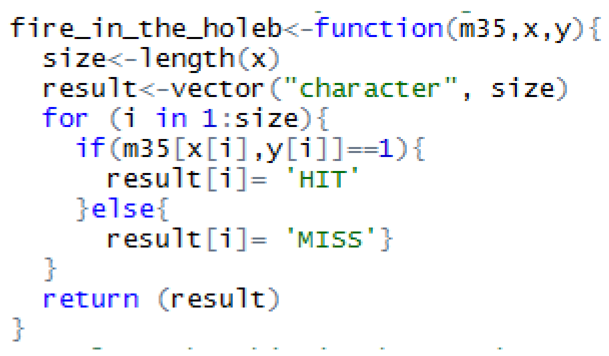


```
insert_shipb <- function(matr,ship_size,pos){
  margen= nrow(matr)-ship_size
  if (pos=='h'){
    row_mat<-sample(1:nrow(matr),1)
    col_mat<-sample(1:margen,1)
    if (check_gapb(matr,row_mat,col_mat,ship_size,pos)){
      col_mat<- col_mat-1                #substract to s
      for (i in 1:ship_size) {
        matr[row_mat,col_mat+i]<-1       #we place our s
      }
    }else {
      matr<-insert_shipb(matr,ship_size,pos)    #if it doe
    }
  }else if (pos=='v'){
    col_mat<-sample(1:ncol(matr),1)
    row_mat<-sample(1:margen,1)
    if (check_gapb(matr,row_mat,col_mat,ship_size,pos)){
      row_mat<- row_mat-1                #substract to s
      for (i in 1:ship_size) {
        matr[row_mat+i,col_mat]<-1       #we place our s
      }
    }else {
      matr<-insert_shipb(matr,ship_size,pos)    #if it doe
    }
  }
  return(matr)
}
```

To introduce the ships in the matrix, we will separate the operations concerning the position value H/V. Then we will generate the random index, we have to take care not to place the ship out of the matrix. Then we will place the ship in a for loop.

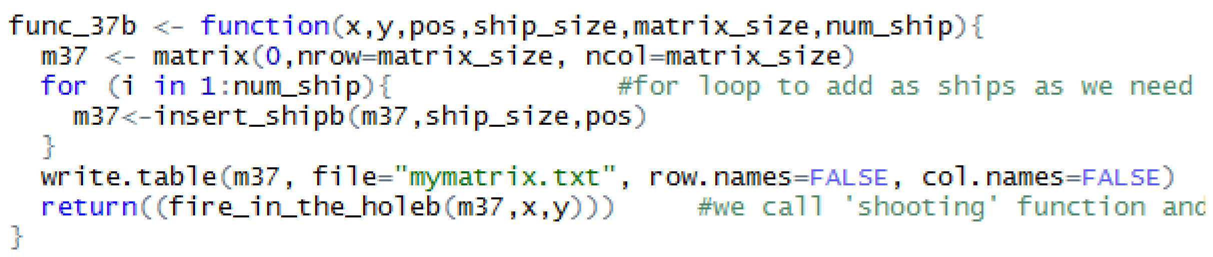


```
check_gapb <- function(matr,row_mat,col_mat,ship_size,pos) {
    if (pos=='h'){
       col_mat<- col_mat-1                         #we  substract 1
        for (i in 1:ship_size) {
          if (matr[row_mat,col_mat+i]==1){          #we check if th
            return (FALSE)                          #FALSE if we CA
          }
        }
    }else if (pos=='v'){
       row_mat<- row_mat-1                         #we  substract 1
        for (i in 1:ship_size) {
          if (matr[row_mat+i,col_mat]==1){          #we check if th
            return (FALSE)                          #FALSE if we CA
          }
        }
    }
  return (TRUE)                                #TRUE if we are allow
}
```



```
fire_in_the_holeb<-function(m35,x,y){
  size<-length(x)
  result<-vector("character", size)
  for (i in 1:size){
    if(m35[x[i],y[i]]==1){
      result[i]= 'HIT'
    }else{
      result[i]= 'MISS'}
  }
  return (result)
}
```

37-



```
func_37b <- function(x,y,pos,ship_size,matrix_size,num_ship){
  m37 <- matrix(0,nrow=matrix_size, ncol=matrix_size)
  for (i in 1:num_ship){                #for loop to add as ships as we need
    m37<-insert_shipb(m37,ship_size,pos)
  }
  write.table(m37, file="mymatrix.txt", row.names=FALSE, col.names=FALSE)
  return((fire_in_the_holeb(m37,x,y)))     #we call 'shooting' function and
}
```

In this case, the function will receive the shoot's coordenates, the position of the ships (vertical or horizontal) and the size of the ships, the size of the matrix and the numbers of ships to place