Lesson I. R language Programming Environment

Ruth Gómez Graciani

2017/10/16

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

```
a <- c(v, 123, v)
a
```

```
[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 varieble

```
my_sqrt <- sqrt(v)
my_sqrt
```

```
[1]  1.772005 11.090537  0.100000
```

4- Now calculate the value of v divided by my_sqrt

```
v / my_sqrt
```

```
[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?

```
c(1, 2, 3, 4) + c(0, 100)
```

```
[1]   1 102   3 104
```

Yes, they can, because the largest vector is a multiple from the smaller one. The small vector is repeated twice in order to add it to the first one.

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

```
1:19
```

```
 [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

```
seq(1, 10, 0.5)
```

```
 [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
[15]  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 n

```
n <- seq(50, 999, length.out = 50)
n
```

```
 [1]  50.00000  69.36735  88.73469 108.10204 127.46939 146.83673 166.20408
 [8] 185.57143 204.93878 224.30612 243.67347 263.04082 282.40816 301.77551
[15] 321.14286 340.51020 359.87755 379.24490 398.61224 417.97959 437.34694
[22] 456.71429 476.08163 495.44898 514.81633 534.18367 553.55102 572.91837
[29] 592.28571 611.65306 631.02041 650.38776 669.75510 689.12245 708.48980
[36] 727.85714 747.22449 766.59184 785.95918 805.32653 824.69388 844.06122
[43] 863.42857 882.79592 902.16327 921.53061 940.89796 960.26531 979.63265
[50] 999.00000
```

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

```
length(n)
```

```
[1] 50
```

The result is the number of elements inside the vector

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

```
b <- list(1:length(n))
b
```

```
[[1]]
 [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] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
[47] 47 48 49 50
```

11- Create a vector with 100 zeros

```
c <- rep(0, 100)
c
```

```
  [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
 [36] 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
 [71] 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
```

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

```
d <- rep(c('a', 'b', 'c'), 10)
d
```

```
 [1] "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b"
[18] "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

```
abc <- c(rep('a', 10), rep('b', 10), rep('c', 10))
abc
```

```
 [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a" "b" "b" "b" "b" "b" "b" "b"
[18] "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

```r
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<1. What is the result of this operation? What kind of value you see?

```r
bool <- v1 < 1
bool
```

```
[1] FALSE FALSE  TRUE FALSE
```

The result of the operation is TRUE when the inequality is accomplished, and FALSE when not, for each value of the vector. The value is a logical operator, as we can see when the `str()` function is used:

```r
str(bool)
```

```
 logi [1:4] 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"

```r
abc <- paste(abc, collapse="")
abc
```

```
[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

```r
e <- c(1:20)
m1 <- matrix(e, 4,5)
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?

```r
names <- c("Ana", "John", "Pedro", "Joan")
m2 <- matrix(c(m1, names), 4, 6)
m2
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "1"  "5"  "9"  "13" "17" "Ana"
[2,] "2"  "6"  "10" "14" "18" "John"
```

```
[3,] "3"  "7"  "11" "15" "19" "Pedro"
[4,] "4"  "8"  "12" "16" "20" "Joan"
```

The numerical values have been converted into characters

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

```r
df <- data.frame(m1, names)
df
```

```
  X1 X2 X3 X4 X5 names
1  1  5  9 13 17   Ana
2  2  6 10 14 18  John
3  3  7 11 15 19 Pedro
4  4  8 12 16 20  Joan
```

The data frame can contain different value types

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

```r
labels <- c("name", "age", "weight",    "bp",    "rate",    "test")
colnames(df) <- labels
df
```

```
  name age weight bp rate  test
1    1   5      9 13   17   Ana
2    2   6     10 14   18  John
3    3   7     11 15   19 Pedro
4    4   8     12 16   20  Joan
```

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

```r
f <- c(10, 11, 12)
mean(f)
```

```
[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.

```r
new_mean <- function(vector) {
  summatory <- sum(vector)
  numofels <- length(vector)
  return(summatory / numofels)
}

new_mean(f)
```

```
[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, 4, replace = T)
```

```
[1] 3 3 6 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?

```
coin <- c('heads', 'tails')
prob <- c(0.7, 0.3)
flip <- sample(coin, 50, prob, replace = T)
table(flip) / 50
```

```
flip
heads tails
 0.62  0.38
```

The `table` function counts the number of elements inside the vector, and dividing it by the total number of elements we obtain the proportion or probability.

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, 150, 10)
```

```
 [1] 155.6031 164.6120 167.8707 154.0799 152.0535 129.6693 177.4067
 [8] 149.2280 151.2564 148.9029
```

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

```
rp <- matrix(nrow = 10, ncol = 100)

for(i in c(1:100)){
  rp[,i] <- rnorm(10, 150, 10)
 }
```

The variable `rp` is partially below:

```
rp[, 1:7]
```

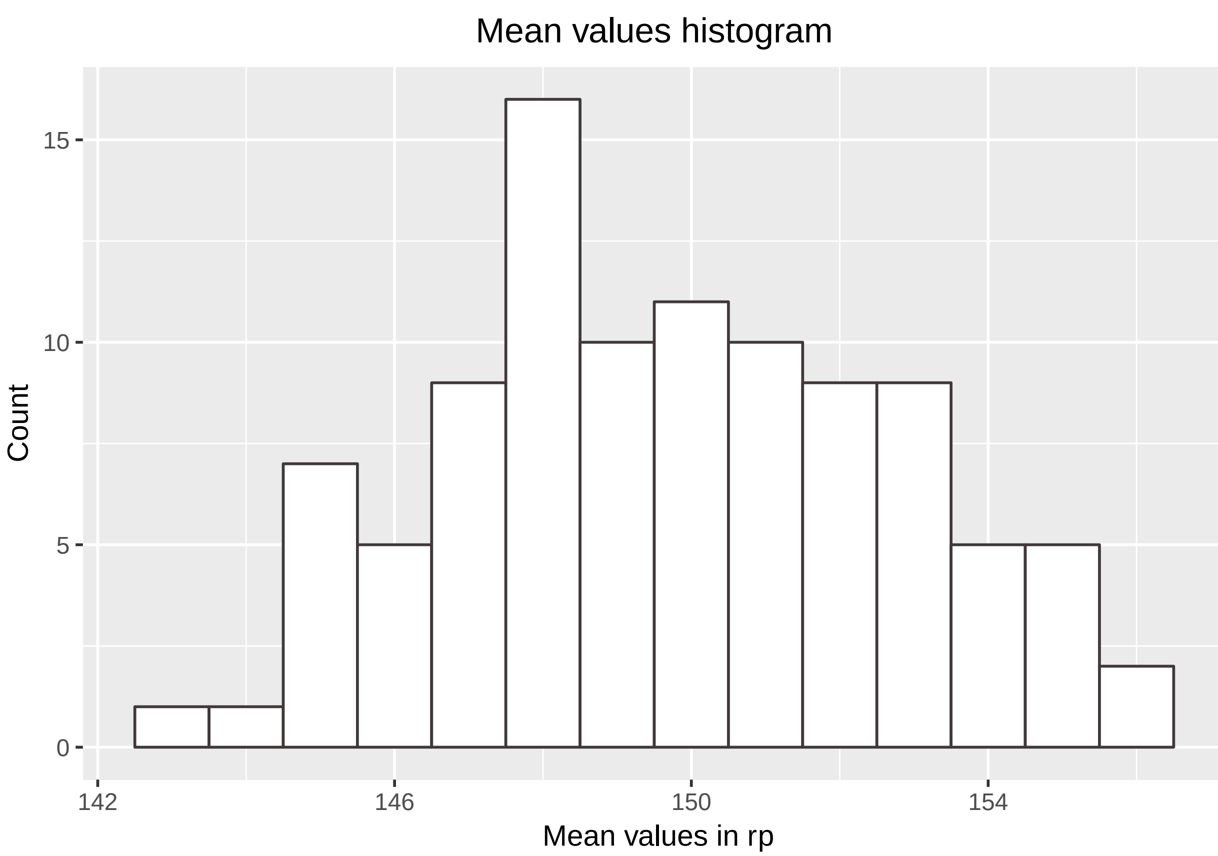
```
          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
 [1,] 140.0669 143.3637 145.1863 152.2051 145.7332 151.4134 158.8728
 [2,] 160.8110 148.3547 138.9019 165.2508 148.3254 147.4687 140.5276
 [3,] 155.5592 146.9632 161.6654 162.8191 149.4475 150.3197 154.3862
 [4,] 158.0649 129.1716 159.2442 136.0860 169.1818 152.7116 139.8331
 [5,] 148.8197 141.4025 138.7841 144.6775 131.1346 173.8950 165.7094
 [6,] 149.0578 144.6640 157.8662 167.9812 163.0143 148.2588 156.8001
 [7,] 142.7917 161.1891 143.7631 154.8161 144.1854 156.8680 158.2100
 [8,] 139.7568 132.0696 168.0300 149.4341 156.7158 166.8079 142.2932
 [9,] 164.4519 151.1789 149.1910 149.7222 151.6728 141.7211 140.8171
[10,] 142.9706 151.1849 137.0063 132.8108 144.3284 159.2679 151.6787
```

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

```
g <- colMeans(rp)
g
```

```
  [1] 150.2351 144.9542 149.9639 151.5803 150.3739 154.8732 150.9128
  [8] 151.9642 146.7104 150.9722 146.7402 146.3249 148.7724 148.3713
 [15] 152.0814 148.0033 152.6465 148.8175 145.7716 153.0002 147.0953
 [22] 148.2386 153.5236 144.4476 148.2192 147.8983 149.1783 148.0680
 [29] 153.6397 152.5251 151.4461 149.8321 152.6800 149.0938 147.6887
 [36] 148.2227 149.7528 149.2105 149.7470 146.8988 148.1423 151.5602
 [43] 152.7502 147.0374 150.8670 154.4379 146.6294 151.0045 151.6000
 [50] 144.8649 144.6366 146.9825 145.1436 145.8679 149.6766 148.3947
 [57] 156.1056 152.6936 151.4451 148.3983 150.1896 145.8903 144.6277
 [64] 146.6376 154.3656 149.2795 151.6863 152.2306 153.3180 148.1821
 [71] 150.8419 155.4791 149.8087 155.3558 149.6408 149.4098 154.0923
 [78] 148.6606 153.0398 150.0385 151.4564 149.2033 150.5751 145.5390
 [85] 145.2656 145.1572 155.5233 151.4097 146.9386 152.2166 147.6251
 [92] 152.2420 148.2120 143.4210 147.7971 148.4928 155.3153 153.3922
 [99] 148.7496 155.1808
```

```
ggplot()+
 aes(g)+
 geom_histogram(binwidth=1, colour="#413839", fill="white")+
 labs(title="Mean values histogram", x = " Mean values in rp", y = "Count" )+
 theme(plot.title = element_text(hjust = 0.5))
```



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

```
X <- sample(100, 10)
X
```

```
 [1] 78 15 92  8 50 83 71 30 10 19
```

```
for (i in c(2:length(X))) {
  X[i] <- X[i] + X[i - 1]
}
X
```

```
 [1]  78  93 185 193 243 326 397 427 437 456
```

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. Is it possible to do the same thing using which()? How would you do it?

With the `for` loop

```
Z1 <- rnorm(20)
Z2 <- rep(0, 20)
for (i in c(1:20)) {
  if (Z1[i] > 0) {
    Z2[i] <- 1
  } else{
    Z2[i] <- -1
  }
}
```

With `which()`

```
Z2 <- rep(0,20)
Z2[which(Z1 > 0)] <- 1
Z2[which(Z1 < 0)] <- -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

```
plants <- read.table("Data/plants2017_10_13D19_4_42 (1).csv", sep = ",")
```

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

As we can see below, there is a large number of rows:

```
nrow(plants)
```

```
[1] 5166
```

```
ncol(plants)
```

```
[1] 10
```

A small subset of the row names will be displayed. All the column names will be displayed

```r
rownames(plants)[1:10]
```

```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

```r
colnames(plants)
```

```
 [1] "Scientific_Name"      "Duration"             "Active_Growth_Period"
 [4] "Foliage_Color"        "pH_Min"               "pH_Max"
 [7] "Precip_Min"           "Precip_Max"           "Shade_Tolerance"
[10] "Temp_Min_F"
```

The size in bytes:

```r
object.size(plants)
```

```
933520 bytes
```

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

```r
head(plants)
```

```
               Scientific_Name          Duration Active_Growth_Period
1                  Abelmoschus              <NA>                 <NA>
2       Abelmoschus esculentus Annual, Perennial                 <NA>
3                        Abies              <NA>                 <NA>
4               Abies balsamea         Perennial    Spring and Summer
5 Abies balsamea var. balsamea         Perennial                 <NA>
6                     Abutilon              <NA>                 <NA>
  Foliage_Color pH_Min pH_Max Precip_Min Precip_Max Shade_Tolerance
1          <NA>     NA     NA         NA         NA            <NA>
2          <NA>     NA     NA         NA         NA            <NA>
3          <NA>     NA     NA         NA         NA            <NA>
4         Green      4      6         13         60        Tolerant
5          <NA>     NA     NA         NA         NA            <NA>
6          <NA>     NA     NA         NA         NA            <NA>
  Temp_Min_F
1         NA
2         NA
3         NA
4        -43
5         NA
6         NA
```

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

`summary()` shows basic statistical values of the vector. In this case, I have assumed that one of the experiment variables is pH_Min.

```r
summary(plants$pH_Min)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  3.000   4.500   5.000   4.997   5.500   7.000    4327
```

NA states fot Not Available (it is missing data) `str()` provides the type of variable and the elements that can be found inside it. As an example, we can see that str is a datagfarme and therefore it can contain different types of values.

```
str(plants, strict.width="cut")
```

```
'data.frame':   5166 obs. of  10 variables:
 $ Scientific_Name     : Factor w/ 5166 levels "Abelmoschus",..: 1 2 3 4 ..
 $ Duration            : Factor w/ 8 levels "Annual","Annual, Biennial",...
 $ Active_Growth_Period: Factor w/ 8 levels "Fall, Winter and Spring",..:..
 $ Foliage_Color       : Factor w/ 6 levels "Dark Green","Gray-Green",..:..
 $ 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..
 $ Temp_Min_F          : int  NA NA NA -43 NA NA NA NA -13 NA ...
```

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

There are already functions to calculate the minimum and average values of a vector, I have assumed that the function has to retrieve both at the same time, from the columns specified, when a datafarme is provided.

```
Calculate_plants <- function(plants){
  minimum <- min(plants$Temp_Min_F, na.rm = TRUE)
  average <- mean(plants$Precip_Min, na.rm = TRUE)
  return(c(minimum, average))
}

Calculate_plants(plants)
```

```
[1] -79.00000  25.56884
```

To be evaluated:

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")

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

The ships can be vertical or horizontal, but they can't overlap.

```
Battleship <- function(X, Y, ships) {
  # Create the empty matrix
  m1 <- matrix(0, 20, 20)
  # Iterate over ships
  for (s in c(1:ships)) {
```

```r
    # We avoid solapping ships
    done <- F
    while (done == F) {
      # Place in a random position, and random direction
      variablenum <- c(1:6) + sample(c(1:14), 1)
      fixnum <- sample(c(1:20), 1)
      random <- sample(c(1:2))
      order <- list()
      order[[random[1]]] <- variablenum
      order[[random[2]]] <- fixnum
      if (sum(m1[order[[1]], order[[2]]]) == 0) {
        m1[order[[1]], order[[2]]] <- c(rep(1, 6))
        done <- T
      }
    }
  }
  # Creates result
  result <- as.factor(diag(m1[X, Y]))
  levels(result) <- c('miss', 'hit')
  # Export matrix and return result
  write.table(m1, 'matrix.txt', col.names = F,row.names = F,quote = F )
  return(as.character(result))
}
```

Here there is an example of the result of the function:

```r
Battleship(c(1, 4, 15, 6, 17, 11), c(10, 14, 16, 2, 7, 20), 3)
```

```
[1] "miss" "miss" "miss" "miss" "miss" "miss"
```

The generated file looks like this (I am using `engine = 'bash'`):

```bash
cat matrix.txt
```

```
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 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 1 1 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
```