A minimalist introduction to R

Timothée Bonnet

November 22, 2018

Contents

1 Math operators

1.1 What you need to know

Below we demonstrate the use of some basic mathematical operators:

```
1+3 #addition
```

```
## [1] 4
```

```
5-2 #substraction
```

```
## [1] 3
```

```
6*4 #multiplication
```

```
## [1] 24
```

```
14/2 #division
```

```
## [1] 7
```

```
2^3 #exponent
```

```
## [1] 8
```

```
2**3 #other version for exponent
```

```
## [1] 8
```

There are many mathematical functions already present in R:

```
exp(3) #exponential
```

```
## [1] 20.08554
```

```
log(2.71) #logarithm
```

```
## [1] 0.9969486
```

```
sqrt(9) #square root
```

```
## [1] 3
```

```
9 ^ (1/2) # other version for square root
```

```
## [1] 3
```

```r
#trigonometric functions :
#(use ";" for several expressions on the same line)
sin(pi/2); cos(1); tan(pi/3)
```

```
## [1] 1
## [1] 0.5403023
## [1] 1.732051
```

1.2 Practice

Use R to compute:

$$y = (\cos(0.1\pi))^3$$

You should get 0.8602387

And:

$$y = \log(3 - 2) + 5$$

You should get 5

And if you dare:

$$y = \frac{1}{2\sqrt{2\pi}} e^{(\frac{3-\pi}{2})^2}$$

You should get 0.2004734

2 Logical operators

2.1 What you need to know

Logical operators are very important for programming and scripting. You can test whether two things are equal with double = signs:

```r
3 == 6/2 #is 3 equal to 6/2? TRUE!
```

```
## [1] TRUE
```

```r
3 == pi  # FALSE!
```

```
## [1] FALSE
```

You can also test if they are NOT equal with the operator `!=`:

```r
2 != 3
```

```
## [1] TRUE
```

```r
2 != 2
```

```
## [1] FALSE
```

The ! symbol means "not" in general, so you can use it to get the opposite result:

```
!TRUE

## [1] FALSE

!FALSE

## [1] TRUE
```

The AND operator is &

```
2 == 2 & 3==3

## [1] TRUE

2 ==2 & 3==2

## [1] FALSE
```

The OR operator is |

```
2 == 2 | 3==2

## [1] TRUE

2 == 4 | 3==2

## [1] FALSE
```

2.2 Practice

Try and guess the result of these logical tests before running them:

```
! 1==2
(1!=2 | 3==4) & (2==4/2)
"abc" != "bc"
```

3 Assignment

3.1 What you need to know

Values can be assigned to objects to store them and make your code flexible. You assign a value to an object using the operator `<-` (or `=`, but be careful not to confuse this with the `==` used in tests).

```r
#You can use objects in calculation
a <- 12
a + 2

## [1] 14

# you can assign an object value to another object
b <- a
c <- a*b

# you can re-assign an object
a <- "c"
b <- "c"
a == b

## [1] TRUE

c <- a == "b"
c

## [1] FALSE
```

4 Containers

4.1 Vectors

The simplest container is a vector. A flexible way to create a vector by *concatenating* several values with the syntax `c(x,y,...)`.

```r
a <- c(3,9,3,5) # c is for concatenate
a

## [1] 3 9 3 5
```

You can now do calculations on your vector:

```r
a * 2

## [1]  6 18  6 10
```

You can access one or several elements in the vector using squared brackets

```r
#access one value
a[1]
```

```
## [1] 3
```

```r
a[2]
```

```
## [1] 9
```

```r
#access multiple values by concatenating locations
a[c(1,3)]
```

```
## [1] 3 3
```

```r
#access mutiple successive values
a[2:4]#the syntax x:y means "all integers between x and y"
```

```
## [1] 9 3 5
```

```r
#modify a value
a[3] <- -5
a
```

```
## [1]  3  9 -5  5
```

```r
#modify mutiple values
a[1:2] <- 1
a
```

```
## [1]  1  1 -5  5
```

Some useful ways to create vectors:

```r
#successive integers:
1:10
```

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

```r
#arbitrary sequence:
seq(from=-0.2, to = 0.5, by=0.03)
```

```
##  [1] -0.20 -0.17 -0.14 -0.11 -0.08 -0.05 -0.02  0.01  0.04  0.07  0.10
## [12]  0.13  0.16  0.19  0.22  0.25  0.28  0.31  0.34  0.37  0.40  0.43
## [23]  0.46  0.49
```

```r
#random numbers:
runif(n = 4, min = 1, max = 5)

## [1] 1.051750 3.229760 4.630275 1.735088
```

4.2 Practice

Create a vector of 100 random numbers between -1 and 1. Then subsitute all values below -0.5 by 0.

Create a sequence of number between 0 and 10. Find a way to repeat the sequence 50 times and save the result to a vector.

5 For-loop

5.1 What you need to know

A for loop follows the basic structure:

```r
  for (i in 1:N)
  {
    something as a function of i0
  }#end of the loop
```

1:N means all the integer values between 1 and 10.
Instead of "i" you can write anything
For instance:

```r
  for (i in 1:10)
  {
    print(i)
  }

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

5.2 Practice

5.2.1 Calculate averages

Load rock data

```
rock <- read.csv("rock.csv")
```

Use a for loop to obtain column averages

5.2.2 Matrix operations

We create two random matrices:

```
Mat1 <- matrix(rnorm(9), nrow = 3)
Mat2 <- matrix(rnorm(9), nrow = 3)
```

We would like to add these two matrices to create third matrix. We initialize it with empty values:

```
MatAdd <- matrix(NA, nrow = 3, ncol = 3)
```

Write a for-loop to fill the cells following $MatAdd_{ij} = Mat1_{ij} + Mat2_{ij}$.
Do the same with two nested for-loops!

Now, we would like to multiply the two matrices. The inner product of matrices V and U to create matrix W is defined by $w_{ij} = \sum_{k=1}^{k=m} v_i k * u_k j$

For instance, the first cell is calculated as:

```
Mat1[1,1]*Mat2[1,1]+Mat1[1,2]*Mat2[2,1]+Mat1[1,3]*Mat2[3,1]

## [1] 1.196214
```

Write nested for-loops to calculate the matrix W.

NB: you can check the result by using the R command **%*%**:

```
Mat1 %*% Mat2
```

6 While-loop

6.1 What you need to know

```
    while(condition TRUE)
    {
      something
    }
```

For instance:

```
x <- 0
while(x<10)
    {
      x <- x+1
      print(x)
    }

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

6.2 Practice

The function sample() takes 5 number between 1 and 6 (like 5 dice!):

```
x <- sample(x = 1:6, size = 5, replace = TRUE)
```

Are all die equal?

```
all(x == x[1])

## [1] FALSE
```

Are they ever going to be equal?
Write a while loop to find a case with all die equal
How many attempts does it take
Write a for while loop within a for loop to estimate how long it take on average.

7 If-else statement

7.1 What you need to know

```
if(condition)
{
  do something
}
```

```
if(condition)
{
  do something
}else{
  do something else
}
```

For instance:

```
for (i in 1:10)
{
  if(i < 6)
  {
    print("tofu")
  }else{
    print("bacon")
  }
}
```

```
## [1] "tofu"
## [1] "tofu"
## [1] "tofu"
## [1] "tofu"
## [1] "tofu"
## [1] "bacon"
## [1] "bacon"
## [1] "bacon"
## [1] "bacon"
## [1] "bacon"
```

7.2 Practice

We can draw 100 random number following a random distribution of mean 0 and variance one with:

```r
x <- rnorm(n = 100, mean = 0, sd = 1)
```

If we take their logarithm we obtain many "NaN" (Not A Number), because the log of a negative number is undefined:

```r
log(x)

## Warning in log(x):  NaNs produced

##   [1]           NaN           NaN           NaN -0.2784549221  0.4460495925
##   [6] -0.7040573757 -0.5724583745           NaN -0.8780464727           NaN
##  [11]           NaN           NaN -1.5879448331           NaN -1.1289670490
##  [16] -0.9526782238 -0.9121447881           NaN -0.0006219726  0.4930575893
##  [21]  0.5428036577           NaN -0.6202018792           NaN -0.8518593389
##  [26]           NaN -0.4724032309 -1.7117975425 -0.9935697376 -0.4283233769
##  [31]           NaN -0.4201106245 -0.7513223156 -1.0548226734 -0.0144097930
##  [36]           NaN           NaN           NaN           NaN -0.4686198408
##  [41]           NaN           NaN -0.2717458362 -1.3837589337           NaN
##  [46]  0.2024773683           NaN           NaN           NaN -1.3159159957
##  [51] -0.3840767051           NaN -0.0054288874           NaN -5.2478312038
##  [56] -4.1740495757 -0.5099744676 -1.3471328905           NaN  0.2519194235
##  [61]           NaN -0.9939852782  0.4579758687           NaN           NaN
##  [66]           NaN           NaN           NaN           NaN           NaN
##  [71]  0.4258811231           NaN  0.4252592222  0.3440896426           NaN
##  [76] -1.3553027243           NaN           NaN           NaN           NaN
##  [81] -0.7819085057  0.6226365624           NaN  0.1717663790 -0.7025232766
##  [86] -0.4436148867 -0.9504400466           NaN -1.6107717662           NaN
##  [91]           NaN           NaN  0.3696680815           NaN           NaN
##  [96]           NaN           NaN           NaN -2.6499721248           NaN
```

Let's say we want 0 instead of NaN.

Use a for loop and an if-else statement to do that.

More difficult: Use a for loop and a while loop to re-draw random numbers until they are all positive.