R Programming 01

Dr Ebin Deni Raj

15/01/2022

This document is prepared exclusively for the first semester Mtech Programme for working professionals.

R Programming Language

Every year, the number of R users grows by about 40%, and an increasing number of organizations are using it in their day-to-day activities. Begin your journey to learn R with us today!We will Take our first steps with R. In this chapter, you will learn how to use the console as a calculator and how to assign variables. You will also get to know the basic data types in R.

For using R **without any** installation:

- Go to https://rstudio.iiitkottayam.ac.in/
- Log in with your **roll number as user name** and use the initial **password send from itsupport** (Check the email send from ITSUPPORT regarding your institute email)

Let's get started.

How it works

In the editor on the R studio type R code to solve the exercises. When you hit the 'enter' button, every line of code is interpreted and executed by R and you get a message whether or not your code was correct. The output of your R code is shown in the console. R makes use of the # sign to add comments, so that you and others can understand what the R code is about. Just like Twitter! Comments are not run as R code, so they will not influence your result. For example, Calculate 3 + 4 in the editor on the right is a comment.

You can also execute R commands straight in the console. This is a good way to experiment with R code, as your submission is not checked for correctness.

```
# Calculate 3 + 4
3 + 4
```

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

```
# Calculate 6 + 12
6+12
```

```
## [1] 18
```

Arithmetic with R

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

Addition: + Subtraction: - Multiplication: * Division: / Exponentiation: ^ Modulo: %% The last two might need some explaining:

The ^ operator raises the number to its left to the power of the number to its right: for example 3^2 is 9. The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 or 5 %% 3 is 2. With this knowledge, follow the instructions below to complete the exercise.

- Type 2^5 in the editor to calculate 2 to the power 5.
- Type 28 %% 6 to calculate 28 modulo 6.

```
# An addition
5 + 5
```

```
## [1] 10
```

```
# A subtraction
5 - 5
```

```
## [1] 0
```

```
# A multiplication
3 * 5
```

```
## [1] 15
```

```
 # A division
(5 + 5) / 2
```

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

```
# Exponentiation
2^5
```

```
## [1] 32
```

```
# Modulo
28 %% 6
```

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

Great! we will head over to the next exercise.

Variable assignment

A basic concept in (statistical) programming is called a variable.

A variable allows you to store a value (e.g. 4) or an object (e.g. a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

">x <-10" The '>' symbol is used tod enote the command prompt and does not need to be typed in.

You can assign a value 4 to a variable my_var with the command

my_var <- 4

```
# Assign the value 42 to x
x <- 42

# Print out the value of the variable x
print(x)
```

```
## [1] 42
```

Have you noticed that R does not print the value of a variable to the console when you did the assignment? x <- 42 did not generate any output, because R assumes that you will be needing this variable in the future. Otherwise you wouldn't have stored the value in a variable in the first place, right?

Suppose you have a fruit basket with five apples. As a data analyst in training, you want to store the number of apples in a variable with the name my_apples.

Type the following code in the editor: my_apples <- 5. This will assign the value 5 to my_apples. Type: my_apples below the second comment. This will print out the value of my_apples.

```
# Assign the value 5 to the variable my_apples
my_apples<-5

# Print out the value of the variable my_apples
print(my_apples)
```

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

Every tasty fruit basket needs oranges, so you decide to add six oranges. As a data analyst, your reflex is to immediately create the variable my_oranges and assign the value 6 to it. Next, you want to calculate how many pieces of fruit you have in total. Since you have given meaningful names to these values, you can now code this in a clear way:

my_apples + my_oranges

- Assign to my_oranges the value 6.
- Add the variables my_apples and my_oranges and have R simply print the result.
- Assign the result of adding my_apples and my_oranges to a new variable my_fruit

```
# Assign a value to the variables my_apples and my_oranges
my_apples <- 5
my_oranges <- 6

# Add these two variables together
print(my_apples + my_oranges)
```

```
## [1] 11
```

```
# Create the variable my_fruit
my_fruit <-my_apples + my_oranges
```

The great advantage of doing calculations with variables is reusability. If you just change my_apples to equal 12 instead of 5 and rerun the script, my_fruit will automatically update as well.

Common knowledge tells you not to add apples and oranges. But hey, that is what you just did, no :-)? The my_apples and my_oranges variables both contained a number in the previous exercise. The + operator works with numeric variables in R. If you really tried to add "apples" and "oranges", and assigned a text value to the variable my_oranges (check in the editor), you would be trying to assign the addition of a numeric and a character variable to the variable my_fruit. This is not possible.

Basic data types in R

R works with numerous data types. Some of the most basic types to get started are:

Decimal values like 4.5 are called numerics. Natural numbers like 4 are called integers. Integers are also numerics. Boolean values (TRUE or FALSE) are called logical. Text (or string) values are called characters. Note how the quotation marks on the right indicate that "some text" is a character.

Lets try some excercise: * Assign my_numeric variable to 42. * my_character variable to “universe”. Note that the quotation marks indicate that “universe” is a character. * my_logical variable to FALSE. * Note that R is case sensitive!

```
# Change my_numeric to be 42
my_numeric <- 42

# Change my_character to be "universe"
my_character <- "universe"

# Change my_logical to be FALSE
my_logical <- FALSE
```

To sum up :

R has 6 basic data types.

- character
- numeric (real or decimal)
- integer
- logical
- complex

Hola !!! What’s that data type?

First try adding 5 + “six”, you will get an error due to a mismatch in data types? You can avoid such embarrassing situations by checking the data type of a variable beforehand. You can do this with the class() function, as the code on the right shows. An example is shown below:

```
# Declare variables of different types
my_numeric <- 42
my_character <- "universe"
my_logical <- FALSE

# Check class of my_numeric
class(my_numeric)
```

```
## [1] "numeric"
```

```
# Check class of my_character
class(my_character)
```

```
## [1] "character"
```

```
# Check class of my_logical
class(my_logical)
```

```
## [1] "logical"
```

R has many data structures. These include

- atomic vector
- list
- matrix
- data frame
- factors

Vector

Vector is a basic data object in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw. A vector's type can be checked with the typeof() function.

Elements of these data types may be combined to form data structures, such as atomic vectors. When we call a vector atomic, we mean that the vector only holds data of a single data type. Below are examples of atomic character vectors, numeric vectors, integer vectors, etc.

- character: "a", "swc"
- numeric: 2, 15.5
- integer: 2L (the L tells R to store this as an integer)
- logical: TRUE, FALSE
- complex: 1+4i (complex numbers with real and imaginary parts) R provides many functions to examine features of vectors and other objects, for example
- **class()** - what kind of object is it (high-level)?
- **typeof()** - what is the object's data type (low-level)?
- **length()** - how long is it? What about two dimensional objects?
- **attributes()** - does it have any metadata?

On your way from rags to riches, you will make extensive use of vectors. Vectors are one-dimension arrays that can hold numeric data, character data, or logical data. In other words, a vector is a simple tool to store data. For example, you can store your daily gains and losses in the casinos. In R, you create a vector with the combine function c(). You place the vector elements separated by a comma between the parentheses. For example:

numeric_vector <- c(1, 2, 3) character_vector <- c("a", "b", "c") Once you have created these vectors in R, you can use them to do calculations.

```
numeric_vector <- c(1, 10, 49)
character_vector <- c("a", "b", "c")

# Complete the code for boolean_vector
boolean_vector <-c(TRUE,FALSE,TRUE)
```

Notice that adding a space behind the commas in the c() function improves the readability of your code.

Playing with vectors

After one week in stock market and still zero gain in your garage, you decide that it is time to start using your data analytical superpowers. Before doing a first analysis, you decide to first collect all the profit and losses for the last week: For stock1_vector: • On Monday you won Rs 140 • Tuesday you lost Rs 50 • Wednesday you won Rs 20 • Thursday you lost Rs 120 • Friday you won Rs 240

For stock2_vector: • On Monday you lost Rs 24 • Tuesday you lost Rs 50 • Wednesday you won Rs 100 • Thursday you lost Rs 350 • Friday you won Rs 10 You only played stock1 and stock2, since there was a delegation of mediums that occupied the craps tables. To be able to use this data in R, you decide to create the variables stock1_vector and stock2_vector.

```
# stock1 from Monday to Friday
stock1_vector <- c(140, -50, 20, -120, 240)
```

```
# Stcok 2  from Monday to Friday
stock2_vector <-c(-24,-50,100,-350,10)
```

To check out the contents of your vectors, remember that you can always simply type the variable in the console and hit Enter. #### Naming a vector As a data analyst, it is important to have a clear view on the data that you are using. Understanding what each element refers to is therefore essential.

```
# Assign days as names of stock_vector
names(stock1_vector) <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
# Assign days as names of stock_vector
names(stock2_vector) <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
```

If you want to become a good statistician, you have to become lazy. (If you are already lazy, chances are high you are one of those exceptional, natural-born statistical talents.)

In the previous exercises you probably experienced that it is boring and frustrating to type and retype information such as the days of the week. However, when you look at it from a higher perspective, there is a more efficient way to do this, namely, to assign the days of the week vector to a variable!

Just like you did with your stock1_vector and stock2_vector, you can also create a variable that contains the days of the week. This way you can use and re-use it.

```
# The variable days_vector
days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

# Assign the names of the day to stcok1_vector and stock2_vector
names(stock1_vector) <-  days_vector
names(stock2_vector) <- days_vector
```

A word of advice: try to avoid code duplication at all times.

Calculating total winnings Now that you have the stock1 and stock2 earnings nicely as named vectors, you can start doing some data analytical magic.

You want to find out the following type of information:

How much has been your overall profit or loss per day of the week? Have you lost money over the week in total? Are you winning/losing money on stock1 or on stock2? To get the answers, you have to do arithmetic calculations on vectors.

It is important to know that if you sum two vectors in R, it takes the element-wise sum. For example, the following three statements are completely equivalent:

c(1, 2, 3) + c(4, 5, 6) c(1 + 4, 2 + 5, 3 + 6) c(5, 7, 9) You can also do the calculations with variables that represent vectors:

a <- c(1, 2, 3) b <- c(4, 5, 6) c <- a + b

```
#Example: Vector sum
A_vector <- c(1, 2, 3)
B_vector <- c(4, 5, 6)

# Take the sum of A_vector and B_vector
total_vector <- A_vector + B_vector

# Print out total_vector
print(total_vector)
```

```
## [1] 5 7 9
```

Now you understand how R does arithmetic with vectors, it is time to get those Ferraris in your garage! First, you need to understand what the overall profit or loss per day of the week was. The total daily profit is the sum of the profit/loss you realized on poker per day, and the profit/loss you realized on stocks per day.

In R, this is just the sum of stock1_vector and stock2_vector

```
total_daily <- stock1_vector+stock2_vector
```

Based on the previous analysis, it looks like you had a mix of good and bad days. This is not what your ego expected, and you wonder if there may be a very tiny chance you have lost money over the week in total? A function that helps you to answer this question is sum(). It calculates the sum of all elements of a vector. For example, to calculate the total amount of money you have lost/won with stock1 you do:

```
# Total profit with stock1
total_stock1 <- sum(stock1_vector)
# Total profit with stock2
total_stock2 <- sum(stock2_vector)
# Total profit overall
total_week <- total_stock1 +total_stock2
# Print out total_week
print(total_week)
```

```
## [1] -84
```

Total Profit? Oops, it seems like you are losing money. Time to rethink and adapt your strategy! This will require some deeper analysis.

```
# Calculate total gains for stock1 and stock2
total_stock1 <- sum(stock1_vector)
total_stock2 <- sum(stock2_vector)
# Check if you realized higher total gains in poker than in roulette
total_stock1 > total_stock2
```

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