Learning R with Kalekye- From once a Beginner’s Perspective

Kalekye

# 1 Session 1

## R Basics

* **R** is an open-source software and a programming language
* **R** runs on Mac OS, Windows and Linux
* Latest version of R can be downloaded from <https://cran.r-project.org>
* Once downloaded, you can install **R** like any other software
* Accept all default settings during installation for simplicity
* The R you have installed is called the **base R**, it has in built functions called packages that can handle tasks like data manipulation and visualization.
* There are specialized packages that can be downloaded and installed in the **base R**.
* R is a language and does not recognize other languages, for instance hello and “hello” are not the same - we shall see later.
* For any other language written in R which is not intended to be recognized as a code, should be preceded by a # (comment), for R not to return an error. For example; #hello
* R is case sensitive; hello and Hello are two different things, be careful!
* If you need help in R precede the function with ?; (eg. ?tidy)
* Once you write any line of code in R click ‘ctrl+enter’ (for windows) and ‘command+enter’ (for MacOS) to run the code
* When naming an object in R use ‘alt+- (windows)’ and ‘option+-’ (for MacOS)to assign ( <- )

## R Studio

* It is a good practice to use an IDE when using R, the most recommended is **Rstudio** which can be downloaded from <https://posit.co/download/rstudio-desktop/>, and install it in the same location where you installed R.
* Install **R** before installing **Rstudio**
* Accept all default settings during installation for simplicity
* **Rstudio** has different panes, but for a beginner, you need the R script, Console, Environment, history, Files, Plots, and Packages.
* **R script**: Where you write and edit R codes. Located in the top-left of the interface.
* **Console**: Where you directly execute R commands and see their output. Located in the bottom-left of the interface.
* **Environment/History**: Shows your current variables and command history. Located in the top-right of the interface.
* **Files/Plots/Packages/Help**: Displays file browser, plots, package management, and help documentation. Located in the bottom-right of the interface
* *Tip: You can customize pane layouts using the “Pane Layout” option in RStudio’s global options menu*

## pre-class quiz

**Instructions:** Answer the questions below by writing short responses or trying them in R. We will discuss them at the beginning of the next class.

1. What is the result of running this code in R?

x <- "hello"  
y <- 1  
z <- c(x, y)

* What is the data type of x and y

1. Try this in R and say the output

typeof(TRUE)  
typeof(3.14)  
typeof("R is fun")  
typeof(c(1, 2, 3))

* Can a vector contain both numbers and characters?

1. What does the mean() function do? Try this

mean(c(5, 10, 15))

* What happens if you run the following instead?

mean(5, 10, 15)

1. R comes with datasets you can explore. Try running this and explain what kind of data you see:

data()  
head(mtcars)

1. Find out what function is used to import a .csv file and an .xlsx file into R.
2. Find out the difference between c(), list(), and data.frame()

# 2 Session 2

* An object is anything that takes a value ([“Cema-Uonbi/Cema\_courses.github.io” n.d.](#ref-zotero-item-2556))
* Objects can be assigned any name except numbers
* To create more than one variable use c(), concatenate

## 2.1 Data types

* In R, variables do not need to be declared with any particular type and can can even change type after they have been set.
* The basic data types in R are numeric, integer, complex, character and logical/Boolean.

### 2.1.1 Numeric

* Any number with or without a decimal.
* Includes doubles and integers
* You can convert any data type to numeric using as.numeric()
* You can also confirm if data is a numeric type by either using is.numeric() function or class() function

numerics <- 10  
numerics\_many <- c(10, 10.5)  
  
is.numeric(numerics\_many)  
class(numerics\_many)

#### 2.1.1.1 Integer

* Natural number, and does not have a decimal.
* Used when creating variable(s) that do not contain decimals eg. counting number of cows in a farm
* Use L for R to differentiate numeric from integers
* You can convert any data type to integer using as.integer()
* To confirm if data is of integer type, either use is.integer() function or class() function

integers <- c(10L, 20L)  
integers  
  
is.integer(integers)

#### 2.1.1.2 Double

* Continuous variables, hallows for decimal
* For example; weight, length, height
* To confirm if data is of double type, either use is.double() function or class() function

doubles <- c(10, 20.1, 6.3)  
doubles  
  
is.double(doubles)  
typeof(doubles)

### 2.1.2 Complex

* Represented as where is the real part, is the imaginary part and is the imaginary unit (sqrt(-1)).
* R on default stays on the real world so you have to use this format for R not to return NaNs
* Use is.complex() or class() to check for complex data type

complexs <- sqrt(-1 + 0i)  
x <- sqrt(-1)

## Warning in sqrt(-1): NaNs produced

is.complex(complexs)

### 2.1.3 Character

* This is textual data
* A vector of one more characters is called a string
* We use (’’) or (““)
* Use typeof(), class() or is.character() to check the data type
* Use as.character() to convert a data to character type
* Use nchar() to find the length of a string
* Use grepl() to check if a character or string is present in the character object of interest
* Use paste() to merge/ concatenate two strings
* Use cat() to introduce line breaks in a sting

characters <- "morning"  
strings <- 'hello world'  
  
typeof(characters)  
typeof(strings)  
  
nchar(characters)  
grepl("o", characters)  
  
paste(characters, strings)  
  
cat("hello\nworld")

### 2.1.4 Boolean/ Logicalvalue

* Value that can either be TRUE or FALSE
* It is an outcome of comparison between two values
* R does the comparison, then if expression is true it will output TRUE and FALSE if the expression is false

x <- 10  
y <- 9  
   
x == y  
  
if (x > y) {  
 print("b is greater than a")  
} else{  
 print("b is not greater than a")  
  
}

### 2.1.5 Factors

* Used to categorize data/ for categorical variables
* Examples include; demographic (eg. male, female)
* Individual code of a factor is called a level
* Use factor() to create
* Use is.factor() or class() to check if data type is a factor
* Use levels() to print the levels
* Use as.factor() to convert variables to a factor

factors <- factor(c("male", "female"))  
  
levels(factors)  
  
factors\_1 <- factor(c("male", "female"), levels = c("male", "female"))  
  
is.factor(factors)   
class(factors)

### 2.1.6 Operators

* For addition we use ‘+’
* For subtraction we use ‘-’
* For multiplication we use ’\*’
* For division we use ‘/’
* For exponent we use ’^ or \*\*’- raises the number to its left to the power of the number to its right
* For modulus ‘%%’ - returns the remainder of the division of the number to the left by the number on the right
* For integer/ to get the whole number of a division without the remainder we use ‘%/%’
* For matrix multiplication we use ’%\*%’ - we shall see this later

a <- 10  
b <- 19  
  
a + b  
a - b  
a \* b  
a / b  
a ^2  
a %% b  
a %/% b

## 2.2 Data Structures

* These are used to store and organize values
* We have different including vectors, lists, matrices, arrays and Dataframes

### 2.2.1 Vectors

* List of items that are of the same type
* Are one dimensional
* Use c(), concatenate function, and separate items using ‘,’
* Use length() to check how many items in a vector
* Use sort() to arrange items in ascending or descending order
* Use [] or [c()] or [c(-1)] to access items in a vector

x3 <- c("hello", "world")  
  
x4 <- c(1,2,4,0.5,6)  
  
x5 <- 1.5:6.5  
  
x6 <- 1.5:6.3  
  
x7 <- seq(0, 100, by = 10)  
  
x8 <- rep(c(1,2,3), each = 3)  
  
x9 <- rep(c(1,2,3), times = 3)  
  
#to find out the how many items we have in a vector, we use length()  
length(x9)  
  
# to arrange items in a vector in either ascending or descending order we use  
# sort()  
  
x10 <- c(5, 6,9,15,21,30,4,23,60,45) # a non-ordered vector  
  
sort(x10, decreasing = FALSE) # ascending order  
  
sort(x10, decreasing = TRUE) # descending order  
  
sort(x10) # the default in R is ascending order  
  
#if you want to access vector items, we use []  
x10[1]  
  
x10[5]  
  
x10[-2] # every item except the second item  
  
x10[-c(1,10)] #every item except the first and the tenth item

### 2.2.2 Lists

* Contains a combination of different ordered data types and is changeable
* Are one dimensional
* We use list() to create lists
* Use length() to check how many items in a list
* Use [] or [c()] or [c(-1)] to access items in a list
* Use %in% to check if an item is present in a list
* use append() to add items in a list
* Use c() to join two or more lists

x11 <- list("apple", 10, 10L, 10.5)  
x11  
  
#to confirm/check the data structure we use class()  
class(x11)  
  
#we can join two or more lists together using c() - concatenate  
  
x12 <- list("apple", 10)  
x13 <- list(10L, 10.5)  
  
x14 <- c(x12, x13)  
x14  
  
# if we want to check if an item is present in a list we use %in%  
  
"apple" %in% x14  
  
"apple" %in% x13  
  
# to add items  
append(x11, "oranges")  
  
append(x11, "oranges", after = 1)

### 2.2.3 Matrices

\_ Two dimensional data that organizes data in a rectangular layout of rows (horizontal) and columns (vertical) - All elements should be of the same data type - Use matrix() to create and specify rows using (nrow) and columns using (ncol) - You can control whether to fill data by rows or columns using (byrow) - Matrix can also be created from lists or vectors - To access items in a matrix use [row, column] - Use cbind() to add columns and rbind() to add rows; elements in a new column or row must be of the same length as of the existing matrix - Use %in% to check for a specific item - Use dim() to check number of rows and columns - Use length() to find dimension of a matrix (rows\*columns)

x15 <- c(1,2,3,4,5,6)  
  
matrix(x15, nrow = 3, ncol = 2)  
matrix(x15, nrow = 3, ncol = 2, byrow = TRUE)  
matrix(x15, nrow = 3, ncol = 2, byrow = FALSE)  
  
x16 <- c(1,2,3,4,5,6)  
  
x17 <- c(8,9,10,11,12,13)  
  
#creating a matrix using three vectors  
matrix(c(x15, x16, x17), ncol = 3)  
  
#creating matrix using two lists  
#Quiz??  
  
# combine three vectors by column, we use cbind()  
#the vectors should be of the same length  
cbind(x15, x16, x17)  
  
# combine three vectors by row, we use rbind()  
#the vectors should be of the same length  
rbind(x15, x16, x17)  
  
thismatrix <- rbind(x17, x16, x15)  
  
#to access an item in a matric we use [] or [c()] for more than one item  
thismatrix[1,2]  
thismatrix[1,]  
thismatrix[,1]  
thismatrix[,c(1,2)]  
  
#checking if an item is in a matrix we use %in%  
12 %in% thismatrix  
50 %in% thismatrix  
  
#check the number of rows and number of columns in a matrix, use dim()  
dim(thismatrix)  
  
#checking the dimension of the matrix, we use length() = rows\*columns  
  
length(thismatrix)

### 2.2.4 Arrays

* Object that can hold multi-dimensional data
* Use array() to create and dim()mto specify dimension
* Elements are of the same data type
* We access elements of an array using []
* Use %in% to check if item is present in an array
* Use dim() to check number of rows and columns
* Use length() to check dimension

arrays <- c(1:14) # one dimension  
  
arrays\_1 <- array(arrays, dim = c(4, 3, 2)) # multi-dimensional

### 2.2.5 Data frames

* Two-dimensional tabular data that stores data in rows and columns, just like an Excel sheet
* Can have different data types, but each column should have the same data type
* Use data.frame() to create
* use [] or [[]] and $ to access columns of a data frame
* Use rbind() to add rows/ add data vertically and cbind() to add columns/ add data horizontally
* Use dim() to check number of rows and columns
* Use ncol()/ length() and nrow() to check numbe of columns or rows respectively

x18 <- c("a", "b", "c", "d", "e", "f")  
  
data\_frame <- data.frame(  
 x15, x16, x17, x18  
)  
  
data\_frame

## x15 x16 x17 x18  
## 1 1 1 8 a  
## 2 2 2 9 b  
## 3 3 3 10 c  
## 4 4 4 11 d  
## 5 5 5 12 e  
## 6 6 6 13 f

summary(data\_frame)

## x15 x16 x17 x18   
## Min. :1.00 Min. :1.00 Min. : 8.00 Length:6   
## 1st Qu.:2.25 1st Qu.:2.25 1st Qu.: 9.25 Class :character   
## Median :3.50 Median :3.50 Median :10.50 Mode :character   
## Mean :3.50 Mean :3.50 Mean :10.50   
## 3rd Qu.:4.75 3rd Qu.:4.75 3rd Qu.:11.75   
## Max. :6.00 Max. :6.00 Max. :13.00

###Session 2 Quiz – Introduction to R {.unnumbered}

**Part A: Multiple Choice (Choose the best answer)**

Which function is used to combine multiple values into a vector?

1. combine()
2. join()
3. c()
4. list()

What data type is this: c(10L, 20L)?

1. Double
2. Integer
3. Numeric
4. Logical

What function would you use to check if a value is numeric?

1. check.numeric()
2. is.numeric()
3. class.numeric()
4. is.num()

Which symbol is used for exponentiation in R?

1. ^
2. \*\*
3. %%
4. Both a and b

What structure allows you to store different data types in one object?

1. Matrix
2. Vector
3. List
4. Data Frame

**Part B: True or False**

paste() is used to combine two strings.

A matrix can contain both characters and numbers.

Factors are used for continuous numerical data.

rep() is used to repeat values in a vector.

You can access a data frame column using $.

**Part C: Short Answers**

Write the R code to create a numeric vector with values 5, 10, and 15.

Which function do you use to:

1. Convert a value to character?
2. Count characters in a string?
3. Check the number of rows and columns in a matrix?

Write a line of R code that checks if the number 12 exists in a matrix called mymatrix.

What is the difference between %% and %/%?

Write the code to create a factor variable with levels “low”, “medium”, and “high”.

### Pre-class Quiz

Research on how to use **R markdown** to write your scripts. Tip: Use the same drop down we used to access **R script**

# References

“Cema-Uonbi/Cema\_courses.github.io.” n.d. https://github.com/cema-uonbi/cema\_courses.github.io. Accessed August 4, 2025.