R Beginners Exercise 1: Basic Syntax

# Introduction

Welcome to R for Beginners Exercise 1! This notebook contains the exercises for the course as well as being a work space for you to use during the session.

To execute a line of code, click on it and press *Ctrl + Enter*.

To execute a chunk of code, click the green run button at the top right corner of the code chunk or highlight the entire code chunk and press *Ctrl + Enter*.

# 1.1 Load packages

Packages provide R with lots of interesting additional features. They can be loaded to R through two methods:

1. Enable it in the Package Window (bottom right) by finding the required package and selecting it.
2. Use the following command to load a package, for example the ggplot2 package:

library(ggplot2)

# 1.2 Printing to console

There are two functions which can be used to print statement(s)/output to console.

1. print() - Commonly used to print simple statement or one variable to console. Does not work well with concatenation\*.
2. paste() - Provides more flexibility as it can be used to print multiple variables to console. Text can be concatenated\* with variable(s).

\*concatenate: Joining two or more variables together as one.

# Assigning values to variables "var1" and "var2"  
var1 = 25  
var2 = 30  
print("Trying to print two variables. ")

## [1] "Trying to print two variables. "

# An error will be returned for this print statement below as print() can only be used to print one variable.   
print(var1, var2)

# This paste statement will print both variables to console.   
paste(var1, var2)

## [1] "25 30"

# Paste statement can also be used to concatenate text with variables.   
paste("Variable 1:", var1, " Variable 2:", var2)

## [1] "Variable 1: 25 Variable 2: 30"

# 1.3 Types of data

## 1.3.1 Basic datatypes

There are 5 basic data types in R:

1. Numeric - Any numbers, including floating-point (decimal) numbers.
2. Integer - Whole numbers only, declared with alphabet “L” after the whole number.
3. Complex - Numbers which involves imaginary number (i).
4. Logical - TRUE or FALSE.
5. Character - A single character or a string (text/sentence), need to be in double quotation mark " ".

For the following variables, assign a value of the matching datatype.

\* Remember to put equal sign for value assignment.

numeric\_data = 2.3  
integer\_data = 10L  
complex\_data = 2 + 1i  
logical\_data = TRUE  
character\_data = "example"  
  
paste(numeric\_data, integer\_data, complex\_data, logical\_data, character\_data)

## [1] "2.3 10 2+1i TRUE example"

## 1.3.2 Data container

### 1.3.2.1 Vector

Multiple values can be stored in R using a vector. Note that all elements in a vector should have the same datatype. If there is a mix of numeric and character values in a vector, numeric variables will be stored as strings.

Vectors are created using the command c(var1, var2, var3), where var1, var2, etc. are variable names. Add your code after the equal signs to create two vectors, one to store numbers and one to store words.

num1 = 1  
num2 = 2  
num3 = 3  
numbers = c(num1, num2, num3)  
print(numbers)

## [1] 1 2 3

txt1 = "apple"  
txt2 = "banana"  
txt3 = "pear"  
words = c(txt1, txt2, txt3)  
print(words)

## [1] "apple" "banana" "pear"

mix = c(num1, num2, txt1, txt2)  
print(mix)

## [1] "1" "2" "apple" "banana"

### 1.3.2.2 Vector without specifying elements

Sometimes it can be useful to create a vector with a specific size without explicitly declaring the variables in it. This can be done by the command below. All the elements within the vector will have a starting value of 0.

emptyContainer = numeric(10)

The vector created using the numeric() command can be used to store variables with a datatype other than numeric. Initially, the vector will have datatype numeric. When a variable with datatype complex or character is stored in it, the vector’s datatype will change. Execute the code chunk below **line-by-line** and monitor the changes to the container’s datatype in the Environment window (top right).

\* To execute a line of code, click on it and press Ctrl + Enter.

emptyContainer = numeric(10) # Vector created with datatype numeric  
emptyContainer[1] = 10.5 # This code will not change the datatype of the vector  
emptyContainer[2] = 3L # This code will not change the datatype of the vector  
emptyContainer[3] = 3+5i # Changes the datatype to complex   
emptyContainer[4] = "apple" # Changes the vector's datatype to character

Notice when the datatype of the vector changes from numeric to complex, the format of all elements in the vector are changed to accommodate a complex number. When the datatype of the vector changes to datatype character, a double quotation mark is added to each element to signify a character type element.

### 1.3.2.3 Table

A data container can also store values in a table-like format. To create a table, combine two or more vectors using the function data.frame() . Each vector will become a column in the table (data-frame).

words\_table = data.frame(numbers, words)  
print(words\_table)

## numbers words  
## 1 1 apple  
## 2 2 banana  
## 3 3 pear

After executing all the code above, variables can be viewed in the Environment window (top right). If you hover your cursor on a variable, the datatype and the byte size of the variable will be shown.

# 1.4 Operators

## 1.4.1 [] Operator

A variable within a data container can be accessed with its index by the [] operator. Note that the index in R starts from 1 and not 0. Complete the print statement below to print the second variable in the mix container.

print(mix[2])

## [1] "2"

It is also possible to access multiple variables in a data container at once. Complete the print statement below to print the first three variables in the mix container.

\* Use the : sign between two indexes in the [] operator to represents “to”. Both the index stated in the square bracket are inclusive.

print(mix[1:3])

## [1] "1" "2" "apple"

Through the index, assign 1 to the 2nd, 3rd, and 4th elements in the emptyContainer.

emptyContainer[2:4] = 1

Now, try printing the following of the words\_table.

\* Use comma (,) to separate the row and column index, e.g., a\_random\_table(row\_index, column\_index.

# The element at the second row and first column  
print(words\_table[2,1])

## [1] 2

# The second row  
print(words\_table[2,])

## numbers words  
## 2 2 banana

# The first column  
print(words\_table[,1])

## [1] 1 2 3

# The last two rows  
print(words\_table[2:3,])

## numbers words  
## 2 2 banana  
## 3 3 pear

Other than specifying the index of a container, the [] operator can also be used to specify condition(s) to choose specific elements in the container.

print(numbers[numbers > 1])

## [1] 2 3

Try using the [] operator to print all the elements in the words container except for “apple”.

\* Use != which stands for not equivalent.

print(words[words != "apple"])

## [1] "banana" "pear"

## 1.4.1 Arithmetic operators

Arithmetic operators are mathematical notations that can be used for calculations. The basic operators are:

* + Addition
* - Subtraction
* \* Multiplication
* / Division
* ^ Exponent
* %% Modulus (Division’s Remainder), e.g., 4 %% 2 = 0

Use the requested arithmetic operators to produce the output required.

# Addition: Create a variable named "ans" which equals to 1 plus 2. Then print "1 plus 2 equals" followed by the value of "ans" to the console.   
ans = 1 + 2  
paste("1 plus 2 equals", ans)

## [1] "1 plus 2 equals 3"

# Subtraction: Create a variable named "ans" which equals to 10 minus 2. Then print "10 minus 2 equals" followed by the value of "ans" to the console.   
ans = 10 - 2  
paste("10 minus 2 equals", ans)

## [1] "10 minus 2 equals 8"

# Multiplication: Create a variable named "ans" which equals to 2 times 4. Then print "2 times 4 equals" followed by the value of "ans" to the console.   
ans = 2 \* 4  
paste("2 times 4 equals", ans)

## [1] "2 times 4 equals 8"

# Division: Create a variable named "ans" which equals to 15 divided by 3. Then print "15 divided by 3 equals" followed by the value of "ans" to the console.   
ans = 15 / 3  
paste("15 divided by 3 equals", ans)

## [1] "15 divided by 3 equals 5"

# Exponential: Create a variable named "ans" which equals to 3 to the power of 2. Then print "3 to the power of 2 equals" followed by the value of "ans" to the console.   
ans = 3 ^ 2  
paste("3 to the power of 2 equals", ans)

## [1] "3 to the power of 2 equals 9"

# Modulus: Create a variable named "ans" which equals to the remainder of 10 divided by 3. Then print "Remainder of 10 divided by 3 equals" followed by the value of "ans" to the console.   
ans = 10 %% 3  
paste("Remainder of 10 divided by 3 equals", ans)

## [1] "Remainder of 10 divided by 3 equals 1"

When doing calculations in R, you can either put your calculations in the source window/code chunk (top left), or in the console window (bottom left). Try using both for the following calculations:

450 / 25

## [1] 18

9.8 \* 35

## [1] 343

(5\*4) / 2

## [1] 10

## 1.4.2 Relational operators

The basic relational operators are:

* == Equivalent
* != Not equivalent
* > Greater than
* < Smaller than
* >= Greater than or equals to
* <= Smaller than or equals to

Complete the statements below with the requested operator so the statement will return TRUE.

test = 10  
  
# equivalent  
test == 10

## [1] TRUE

# not equivalent  
test != 5

## [1] TRUE

# greater than   
test > 5

## [1] TRUE

# lesser than   
test < 20

## [1] TRUE

# greater or equals to   
test >= 10

## [1] TRUE

# lesser or equals to   
test <= 100

## [1] TRUE

## 1.4.3 Logical operators

There are three basic types of logical operators:

1. & - AND (TRUE when all conditions are satisfied)
2. | - OR (TRUE when at least one of the conditions is satisfied)
3. ! - NOT (TRUE when the condition is not satisfied)

Write code to represent the situations described, using a logical operator of your choice.

\* Remember to use == which stands for ‘is equivalent to’

# 10 is divisible by both 5 and 2  
10%%2==0 & 10%%5==0

## [1] TRUE

# num is either 1 or 5  
num = 5  
num==1 | num==5

## [1] TRUE

# num is not 10  
!(num == 10)

## [1] TRUE