ECO 274 LAB: R Syntax and Data Structure

Learning Objectives

- To understand the logical comparison
- Describe and utilize built-in functions in R.
- Modify default behavior of functions using arguments in R.
- Demonstrate how to create user-defined functions in R
- To familiarize with R- package and libraries

Logical Comparison

Logical operators include greater than (>), less than (<), and equal to (==). A full list of logical operators in R is displayed below:

Operator	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
&	and
|	or

```
3 < 4
3 > 4
#Contrast with 3 = 4; see section about variables below
3 == 4
#!= means "not equal to"
3 != 4
4 >= 5
4 <= 5
2 + 2 == 5
10 - 6 == 4
```

Basic/Built-in functions

What are functions?

A key feature of R is functions. Functions are “self contained” modules of code that accomplish a specific task. Functions usually take in some sort of data structure (value, vector, dataframe etc.), process it, and return a result. The general usage for a function is the name of the function followed by parentheses:

function_name(input)

The input(s) are called arguments, which can include:

```
sqrt(81)
round(3.14159)
round(3.14159, 2)
```

The basic (built-in) functions are available as part of R’s built in capabilities, and we will explore a few more of these base functions below.

```
my_vec_<- c(88, 95, 92, 97, 96, 97, 94, 86, 91, 95, 97, 88, 85, 76, 68)
mean(my_vec)
median(my_vec)
sd(my_vec)
hist(my_vec)
range(my_vec)
sum(my_vec)
summary(my_vec)

p_value <- .034/15476587634566
p_value
```

Built-in Functions in R

There are plenty of helpful built-in functions in R used for various purposes. Some of the most popular ones are:

- min(), max(), mean(), median() – return the minimum / maximum / mean / median value of a numeric vector, correspondingly
- sum() – returns the sum of a numeric vector
- range() – returns the minimum and maximum values of a numeric vector
- abs() – returns the absulute value of a number
- str() – shows the structure of an R object
- print() – displays an R object on the console
- ncol() – returns the number of columns of a matrix or a dataframe

- length() – returns the number of items in an R object (a vector, a list, etc.)
- nchar() – returns the number of characters in a character object
- sort() – sorts a vector in ascending or descending (decreasing=TRUE) order
- exists() – returns TRUE or FALSE depending on whether or not a variable is defined in the R environment

User-defined Functions

One of the best ways to improve your reach as a data scientist is to write functions. Functions allow you to automate common tasks in a more powerful and general way than copy-and-pasting. Writing a function has three big advantages over using copy-and-paste:

- You can give a function an evocative name that makes your code easier to understand.
- As requirements change, you only need to update code in one place, instead of many.
- You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

Writing good functions is a lifetime journey. Even after using R for many years, you still learn new techniques and better ways of approaching old problems.

Function Components

The different parts of a function are −

Function Name − This is the actual name of the function. It is stored in R environment as an object with this name.

Arguments − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

Function Body − The function body contains a collection of statements that defines what the function does.

Return Value − The return value of a function is the last expression in the function body to be evaluated.

The structure of a function is given below:

```
name_of_function <- function(argument1, argument2) {
    statements or code that does something
    return(something)
}
```

When defining the function you will want to provide the list of arguments required (inputs and/or options to modify behavior of the function), and wrapped between curly brackets place the tasks that are being executed on/using those arguments. The argument(s) can be any type of object (like

a scalar, a matrix, a dataframe, a vector, a logical, etc), and it's not necessary to define what it is in any way.

Finally, you can "return" the value of the object from the function, meaning pass the value of it into the global environment. The important idea behind functions is that objects that are created within the function are local to the environment of the function – they don't exist outside of the function.

Let's try creating a simple example function. This function will take in a numeric value as input, and return the squared value.

```
square_it <- function(x) {
    square = x * x
    return(square)
}

square_it(5)
```

```
my_love <- function(myLove = "Disney") {
  paste("I like to visit ", myLove)
}

my_love("Niagara Falls")
my_love("Statue of Liberty")
my_love() # will get the default value, which is Disney
my_love("The Grand Canyon")
my_love("Chicago")
```

Let's try more:

```
fahr_to_cel <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(temp_C)
}

fahr_to_cel(32)
fahr_to_cel(78)
```

```
new.function_1 <- function(a) {
  for(i in 1:a) {
    b <- i^2
    print(b)
  }
}

# Call the function new.function supplying 6 as an argument.
new.function_1(6)
```

Also,

```
# Create a function with arguments.
new.function_2 <- function(a,b,c) {
  result <- a * b + c
  print(result)
}

# Call the function by position of arguments.
new.function_2(5,3,11)

# Call the function by names of the arguments.
new.function_2(a = 11, b = 5, c = 3)
```

```
mean_median <- function(vector){
    mean <- mean(vector)
    median <- median(vector)
    return(c(mean, median))
}
print(mean_median(c(1, 1, 1, 2, 3)))
```

```
add_or_subtract <- function(first_num, second_num, type = "add") {

  if (type == "add") {
    first_num + second_num
  } else if (type == "subtract") {
    first_num - second_num
  } else {
    stop("Please choose `add` or `subtract` as the type.")
  }

}


add_or_subtract(10, 6, type = "add")

add_or_subtract(5, 6, type = "subtract")

add_or_subtract(5, 6, type = "multiply")
```

Packages and Libraries

Packages are collections of R functions, data, and compiled code in a well-defined format, created to add specific functionality. Currently, the CRAN package repository features **18552** available packages and growing. You may check the number of packages currently available on the CRAN.

```
Sys.Date()

nrow(available.packages())
```

The directories in R where the packages are stored are called the libraries. The terms package and library are sometimes used synonymously and there has been discussion amongst the community to resolve this. There are many packages that are available in CRAN and Bioconductor, but there are also packages that are specific to one repository.

Package installation from CRAN

Packages for R can be installed from the CRAN package repository using the install.packages function. This function will download the source code from on the CRAN mirrors and install the package (and any dependencies) locally on your computer. An example is given below for the ***moments*** package that will be required to calculate skewness and kurtosis. Run this code to install ***moments***.

```
install.packages("moments")
```

Loading libraries

Once you have the package installed, you can load the library into your R session for use. This is your first time using moments, how do you know where to start and what functions are available to you? One way to do this, is by using the Package tab in RStudio.

```
library(moments)
skewness(my_vec)
kurtosis(my_vec)

##A few most popular and widely used R-packages in data science

tidyverse
dplyr
ggplot2
```

```
data.table
tidyr
Shiny
plotly
knitr
mlr3

##Project/Book specific package:
#Introduction to Computational Finance and Financial Econometrics

install.packages("IntroCompFinR", repos="http://R-Forge.R-project.org")
```

Reference and acknowledgement:

1. The materials used in this lesson are adapted from work that is Copyright © Data Carpentry (http://datacarpentry.org/), data camp, data quest, Kaggle, and Harvard Chan Bioinformatics Core (HBC) under the open access terms of the Creative Commons Attribution license (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

2. The Book of R: A First Course in Programming and Statistics by Tilman M. Davies