Introduction to R

You can open an R script in RStudio by going to File → New File → R script in the menu bar. RStudio will then open a fresh script above your console pane

```
#Get your current working directory
getwd() #hit Run or press Ctrl+Enter to run this line
```

```
#Create a new directory within the working directory
dir.create("Topic 1_Introuduction to R")
```

```
#List the files in the working directory
dir()
list.files()
```

Arithmetic in R

```
#Just type in the entire formula
5 + 6
9 - 3
4 * 8
12 / 3
2^4
(5 + 9)/2 + 12^2
sqrt(9) #sqrt is a function calculates the square root
log(10)
log2(4)
log(9, base = 3) #set the base
```

Assign value and name to a R object/Variable

```
x = 5
x

y <- 10; y #We can state more than one commands in one line by ";"

n = "black"; print(n)

sumN <- 5 + 1 * 4
print(sumN)
```

Data Types in R

Vector

A vector is a data structure, which can be constructed using the c() function, and assigned to a named object using the = or ← operator.

```
vec = c(1,2,3,7,8,9,10,14)
vec
vec[1:3]
vec[4:8]

length(vec) #the number of elements in vector

rm(vec) #remove the variable

vec1 = c(10:20)
vec1

vec2 = seq(1,10,1)
vec2

vec3 = seq(10,20,2)
vec3

vec4 = seq(1,2, by = 0.1)
vec4

rm(vec1, vec2, vec3, vec4)

color = c("red","blue","yellow","green","purple")
#Reverse the elements
rev(color)
color[c(1,3,5)] #indicate the elements by positions

color1 = rep("red", 8)
color1

color2 = c(c("bule","green"), rep("red",2), rep("purple",3))
color2

#days
days_vector = c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
days_vector[3]
days_vector[c(3,5)]
days_vector[3:5]
rev(days_vector)

#A logical vector
l = c(TRUE, TRUE, FALSE, T, F) #note, no quotes needed
print(l) #print the vector
```

```
#Vector calculation
a = c(1,2,3,4,5)
a * 2
a * a
```

```
b = c(7,3,9,3,4)
a + b
```

```
#Try this
b = c(6,5,4,7,8,9)
a + b
```

```
 #list all the variables in your workspace
ls()

#remove all the variable in your workspace
rm(list = ls())
```

Matrix

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
my_matrix = matrix(1:9, byrow = TRUE, nrow =3)
my_matrix

my_matrix = matrix(1:9, byrow = FALSE, nrow =3)
my_matrix

my_matrix = matrix(1:20, byrow = FALSE, nrow =4)
my_matrix
```

```
#Transposition of the matrix m
m1 = t(my_matrix)
m1

#nameing rows and columns
rownames(my_matrix) = c("r1","r2","r3","r4")
colnames(my_matrix) = c("c1","c2","c3","c4","c5")
my_matrix
```

```
#Sums by row
rowSums(my_matrix)

#Sums by column
colSums(my_matrix)
```

An example, box office Star Wars (in millions!)

```
new_hope <- c(460.998, 314.4)
empire_strikes <- c(290.475, 247.900)
return_jedi <- c(309.306, 165.800)

#Create a matrix from vectors
box_office <- c(new_hope, empire_strikes, return_jedi)

star_war <- matrix(box_office, nrow = 3, byrow = TRUE)
star_war

title <- c("new_hope", "empire_strikes", "return_jedi")
title

region <- c("US", "non-US")
region

rownames(star_war) = title
colnames(star_war) = region
```

Display the matrix star_war

```
> star_war
                    US non-US
new_hope       460.998  314.4
empire_strikes 290.475  247.9
return_jedi    309.306  165.8
```

Alternatively, use **rbind** function which combines vector, matrix or data frame by columns

```
star_war <- rbind(new_hope, empire_strikes, return_jedi)
star_war

colnames(star_war) = region

star_war

rowSums(star_war)

colSums(star_war)
```

```
#selected elements of the matrix
star_war[1,1]

#select the entire second row
star_war[2,]

#select a subset defined by row and column number
star_war[1:2, 1:2]
```

Factor

Factors are the data objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector.

Factors are created using the **factor()** function. You can see the possible levels for a factor through the **levels()** command.

```
#Create a vector contains the observations that belong to a limited number of categories
gender_vector = c("Male", "Female", "Female", "Male", "Male")

#Create a factor objective
factor_gender_vector = factor(gender_vector)
factor_gender_vector

levels(factor_gender_vector)

summary(factor_gender_vector)

table (factor_gender_vector)
```

Data frame

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
name <- c("John", "Mary", "David", "Joe")
gender <- c("male", "female", "male", "female")
married <- c(TRUE, TRUE, FALSE, FALSE)
age <- c(42, 28, 19, 10)

info_df <- data.frame(name, gender, married, age)
info_df

info_df$name
info_df$married
```

List

List is the R object which contain elements of different types numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

```
#Create a list
info_list = list(name, gender, married, age)
info_list

class(info_list)
```

```
#The list allows different lengths of vectors
x = c(1,2,3,4,5,6)

info_list_1 = c(info_list, list(x))
info_list_1
```

Convert Data Type

```
star_war_DF = as.data.frame(star_war)
class(star_war_DF)

info_matrix = as.matrix(info_df)
info_matrix
class(info_matrix)
```

Control Structures

These allow you to control the flow of execution of a script typically inside of a function. Common ones include:

. **if, else**

. **for**

. **while**

. **repeat**

. **break**

. **next**

. **return**

Loop

R programming language provides the following kinds of loop to handle looping requirements.

- **repeat loop**: Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable
- **while loop**: Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body
- **for loop**: Like a while statement, except that it tests the condition at the end of the loop body.

Loop Control Statements

Loop control statements change execution from its normal sequence. R supports the following control statements.

- **break statement**: Terminates the loop statement and transfers execution to the statement immediately following the loop.
- **next statement**: The next statement simulates the behavior of R switch.

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

```
color = c("red","yellow","blue","purple","black")
for(col in color){
  print(col)
}
```

```
color = c("red","yellow","blue","purple","black")
for(col in color){
  if(col != "blue"){
    print(col)
  }
}
```

```
for(j in 1:20){
  if(j > 10){
    print(j)
  }else{
    print(j * 10)
  }
}
```

Function

A function is a set of statements organized together to perform a specific task. R has a large number of build-in functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

An R function is created by using the keyword **function**.

R has many build-in functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as user defined functions.

Build-in Function

```
# Find mean of numbers from 10 to 35 using build-in function mean()
print(mean(10:35))

# Find sum of numbers from 51 to 79 using build-in function sum()
print(sum(51:79))
```

User-defined Function

Create a user defined function, **calcu()**, to calculation the operations using two variables

```
calcu = function(a, b){
  result = (a + b * 5)^2
  return(result)
}
```

Calling the function **calcu()**

```
#Calling the calcu function supplying 4 and 2 as arguments
calcu(4,2)

#Calling the calcu function supplying 4 and 2 as arguments
calcu(0.5, 0.1)
```