R – Programming Introduction – Part 2

Agenda of this activity

In this activity, we are going to learn the below aspects of R Programming

1. Loops
2. Functions
3. Matrices
4. Matrices – Deep dive
5. Lists
6. Thought Activity

1. Loops

Loops are used in programming to repeat a specific block of code.

For Loop:

The for loop in R is used to iterate over a vector and repeat a set of instructions (or a single instruction)

Syntax:

```
for (val in sequence) {

statement

}
```

Example: In the below example, we create a vector and iterate through the vector to print only the values that are even.

```
# create a vector
x = c(2, 5, 3, 9, 8, 11, 6)

# initialize a count variable
count = 0

# A for loop to iterate over the vector x and count the
number of even numbers
for (val in x) {
  if(val %% 2 == 0) {
    count = count+1
  }
}

# print the count
```

```
count
```

<u>While loop:</u>

while loops are used to loop until a specific condition is met.

<u>Syntax:</u>

```
while (test_expression) {

statement

}
```

<u>Example: In the below example, we initialize a variable ‘i’ with 1 and increment it by 1 in a while loop. Every time we increment i, we will print the value of i.</u>

```
# set i to 1
i = 1

# run a 'while' loop till i is less than 5(from 1 to 5) and
print i
while (i < 6) {
  print(i)
  i = i+1
}
```

<u>‘Break’ Statement:</u>
A break statement is used inside a loop to stop the iterations and flow the control outside of the loop.

<u>Example: In the below example, the loop will be broken when the iterating variable ‘val’ reaches a value of 3</u>

```
# create an vector from 1 to 5
x = 1:5

# break the for loop when it encounters a value of 3
for (val in x) {
  if (val == 3){
    break
  }
  print(val)
}
```

<u>'Next' statement:</u>
A next statement is useful when we want to skip the current iteration of a loop without terminating it.

<u>Example: In the below example, only one instance of the loop will be skipped from execution when iterator value reaches 3.</u>

```
# create an vector from 1 to 5

x = 1:5

# skip  the for loop only when it encounters a value of 3
for (val in x) {
  if (val == 3){
    next
  }
  print(val)
}
```

<u>Activity</u>

1. Create a vector with numbers from 1 to 10. Now, initialize a for loop for that vector:
    1. If the number is even , print 'Even Number'
    2. If the number is odd, print 'Odd Number'
2. Write a for loop that iterates over the numbers 1 to 7 and prints the cube of each number using print().

3. Functions

The main aim of functions is to group a small set of re-usable instructions that perform a particular task.

This helps in bringing readability and efficiency to the code.

Syntax:

```
func_name <- function (argument) {

statement

}
```

<u>Example: Function to take x and raise it by power of y.</u>

```
 # Function Definition
# Default Values for y Arguments
pow <- function(x, y = 3) {
  result <- x^y
  print(paste(x, "raised to the power", y, "is", result))
}

# Call a function
pow(8, 2)

pow(3)
```

<u>Example2: Function to print the sign of a number.</u>

```
# R Return Value from Function
check = function(x) {
  if (x > 0) {
    result <- "Positive"
  }
  else if (x < 0) {
    result <- "Negative"
  }
  else {
    result <- "Zero"
  }
  return(result)
}

check(1)
```

4. Matrices

Matrices are tabular data structures that store data of a single type
Matrices can be constructed using the "matrix()" function

Example-1: Create empty matrix

```
# create a matrix
m<- matrix(nrow = 2, ncol = 3)
m
# display the dimensions of the matrix (a 2 x 3 matrix in
this case)
dim(m)
```

Example-2: Create matrix using a vector (observe how the matrix is created)

```
m1 <- matrix(1:6, nrow = 2, ncol = 3)   # here byrow=FALSE
m1
```

Example-3: Create matrix using a vector (filling the rows first)

```
# Matrices can be constructed row-wise also by defining
parameter 'byrow'
m2 <- matrix(1:6, nrow = 2, ncol = 3, byrow=TRUE)
m2
```

Example-4: converting vector datatype to matrix

```
m <- 1:10
m
class(m)
dim(m) <- c(2, 5)
m
```

cbind-ing and rbind-ing :
Matrices can also be created by combining multiple vectors columnwise(using cbind) or rowwise(using rbind).

Example: Combining 2 vectors to create a matrix

```
# Constructing Matrices by combining vectors using cbind and
rbind
x <- 1:3
y <- 10:12

cbind(x, y)
rbind(x, y)
```

Giving Dimension names
Names (headings) can be assigned to rows and columns of a matrix in two ways. One way is by passing the names in the 'dimnames' parameter when creating the matrix and the other way is by assigning to 'colnames' and 'rownames' of the matrix after the matrix is created.

Example : Adding column names and row names to a Dataframe

```
 x = matrix(1:9, nrow = 3,
           dimnames = list(c("X","Y","Z"),
                           c("A","B","C")))
dim(x)
colnames(x)
rownames(x)
class(x)
```

An Alternate way:

```
colnames(x) = c("C1", "C2", "C3")

rownames(x) = c("R1", "R2", "R3")
```

5. Matrices – Deep dive

Sub-setting elements of a Matrix

Since a matrix has two dimensions, we need two positional indices to subset a particular element of a matrix
The first positional index is for the row and the second positional index is for the column

Example:

```
# Matrices can be subset in the usual way with (i , j) type
indices.
#Create a matrix
x <- matrix(1:6, 2, 3)
x
# Refer to the 1st row, 2nd element
x[1, 2]
# Refer to the 2nd row, 1st  element
x[2, 1]
```

```
# row/column  Indices can also be excluded as cited below to
get only columns or only rows respectively
x[1, ]

x[, 2]
```

Matrix Operations

A few operators for matrices are as follows:

"+" --> Matrix addition

"-" --> Element wise subtraction

"*" --> Element wise multiplication

"%*%" --> Matrix Multiplication

Create two matrices to perform matrix operations:

```
matrix_one <- matrix(c(1:9), nrow = 3, ncol = 3)

matrix_two <- matrix(c(10:18), nrow = 3, ncol = 3)

matrix_one

matrix_two
```

All Matrix operations:

```
## element-wise addition
matrix_two + matrix_one

## element-wise subtraction
matrix_two - matrix_one

## element-wise multiplication
matrix_two * matrix_one

## true matrix multiplication
matrix_two %*% matrix_one
```

Activity

1. Create a matrix with 4 rows and 3 columns having Months of the year e.g. "Jan", "Feb", etc. as values
2. Create a matrix by combining following vectors using rbind and cbind -
   - vector_sub1 <- c(1, 2, 3, 4)
   - vector_sub2 <- c(8, 16, 24, 32)
3. Create a matrix A with numbers 1 to 12, of dimensions 3 rows and 4 columns [numbers should be assigned row-wise]
4. Create a vector with from 13 to 24. Assign this vector to a matrix in the shape 4x3.
5. Now , multiply the 2 matrices obtained from question 3 and question 4
6. Now, in the resultant matrix extract only the second row

```
# 1)

mon<-
c("Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct"
,"Nov","Dec")
mon
mat<-matrix(mon, nrow=4, ncol=3, byrow = TRUE)
mat

# 2)

vector_sub1 <- c(1, 2, 3, 4)
vector_sub2 <- c(8, 16, 24, 32)

mat_rbind<-rbind(vector_sub1,vector_sub2)
mat_rbind

mat_cbind<-cbind(vector_sub1,vector_sub2)
mat_cbind
```

7. Lists

A list is an object consisting of an ordered collection of objects known as its components. Each component in a list can be of any datatype. For example, a list could consist of a numeric vector, a logical value and a matrix.

Example: Creating a List

```
# create a list with below syntax
first_list <- list(a = c(1, 2, 3), b = TRUE, c = 20)
first_list
```

`Accessing components in a list with index`

```
# To refer to a particular element in a list the index can be
used similar to a vector.
first_list[1]
first_list[2]
first_list[3]
```

<u>Accessing components in a list with key</u>

```
# Also, a particular component can be referred using it’s key
first_list$a
first_list$b
first_list$c
```

<u>Modifying components in a list:</u>

```
first_list[[‘b’]] = FALSE
```

<u>Deleting a component:</u>

```
first_list[[‘c’]] = NULL
```

<u>Activity</u>

1. Create a list containing strings, numbers, vectors and a logical value.
2. Create a list containing a vector, a matrix and a list and give names to the elements in the list
3. create a list containing a vector, a matrix and a list and add element at the end of the list.
4. In the above list, select second element
5. In Q3, select third element of the list in the nested list

6. Thought Activity

Try representing the data in the below paragraph in the best way possible:

"

India has an enormous potential for investment due to its thriving economy. The investors plan for setting up the retail stores in city of Bangalore. After preliminary analysis they found that, the potential localities are HSR Layout, Koramangala, BTM and Murgeshpalya. The price of the land in HSR layout is Rs.14000/sqft with the population density of 4095/sqkm ,The cost of the land soared high in Korgmangala with Rs. 20000/sqft. The population density of Koramangala is 5009/sqkm. The cost of the land in BTM was in par with HSR layout with 13500/sqft and population density of 3725/sqkm. The cost of land in Murgeshpalya was the least with Rs. 10000/sqft and population density of 4700/sqkm. These localaities were categorized as high(>15000), medium(11000-15000) and low(<=10000) based on the cost of the land.

"

Now, is there a data type that suits to this data representation perfectly???