Master the basics of R Programming

Introduction

R is a programming language created and developed in 1991 by two statisticians at the University of Auckland, in New Zealand. It officially became free and open-source only in 1995. For its origins, it provides statistical and graphical techniques, linear and non-linear models, techniques for time series, and many other functionalities. Even if Python is the most common in the Data Science field, R is still widely used for specialized purposes, like in financial companies, research, and healthcare.

Assignment

When we program in R, the entities we work with are called objects [1]. They can be numbers, strings, vectors, matrices, arrays, functions. So, any generic data structure is an object. The assignment operator is `<-`, which combines the characters `<` and `-`. We can visualize the output of the object by calling it:

```
# Assignment
x <- 23
```

A more complex example can be:

```
# A more complex example
x <- 1/1+1*1
y <- x^4
z <- sqrt(y)
x
```

```
## [1] 2
```

```
y
```

```
## [1] 16
```

```
z
```

```
## [1] 4
```

As you can notice, the mathematical operators are the ones you use for the calculator on the computer, so you don't need the effort to remember them. There are also mathematical functions available, like `sqrt`, `abs`, `sin`, `cos`, `tan`, `exp`, and `log`.

Vectors in R Programming

In R, the vectors constitute the simplest data structure. The elements within the vector are all of the same types. To create a vector, we only need the function `c()`:

```
# Create vector
v1 <- c(2,4,6,8)
v1
```

```
## [1] 2 4 6 8
```

This function simply concatenates different entities into a vector. There are other ways to create a vector, depending on the purpose. For example, we can be interested in creating a list of consecutive numbers and we don't want to specify them manually. In this case, the syntax is `a:b` , where a and b correspond to the lower and upper extremes of this succession. The same result can be obtained using the function `seq()`

```
# Creating a list of consecutive numbers
1:7
```

```
## [1] 1 2 3 4 5 6 7
```

The function `seq()` can also be applied to create more complex sequences. For example, we can add the argument by the step size and the length of the sequence:

```
# Create list by step size
v4 <- seq(0,1,by=0.1)
v4
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
# Create list by the length of the sequence
v5 <- seq(0,2,len=11)
v5
```

```
##  [1] 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0
```

To repeat the same number more times into a vector, the function rep() can be used:

```
# Repeat the same number more times into a vector
v6 <- rep(2,3)
v6
```

```
## [1] 2 2 2
```

```
v7 <- c(1,rep(2,3),3)
v7
```

```
## [1] 1 2 2 2 3
```

There are not only numerical vectors. There are also logical vectors and character vectors:

```r
# Logical vector
x <- 1:10
y <- 1:5
l <- x==y
l
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```r
# Character vector
c <- c('a','b','c')
c
```

```
## [1] "a" "b" "c"
```

factors in R Programming

factors are specialized vectors used to group elements into categories. There are two types of factors: ordered and unordered. For example, we have the countries of five friends. We can create a factor using the function `factor()`

```r
# Create a factor
states <- c('italy','france','germany','germany','germany')
statesf <- factor(states)
statesf
```

```
## [1] italy   france  germany germany germany
## Levels: france germany italy
```

To check the levels of the factor, the function `levels()` can be applied.

```r
# Check the levels of the factor
levels(statesf)
```

```
## [1] "france"  "germany" "italy"
```

Matrices in R Programming

As you probably know, the matrix is a 2-dimensional array of numbers. It can be built using the function `matrix()`

```r
# Creating a matrix
m1 <- matrix(1:6,nrow=3)
m1
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

```r
m2 <- matrix(1:6,ncol=3)
m2
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

It can also be interesting combine different vectors into a matrix row-wise or column-wise. This is possible with `rbind()` and `cbind()`:

```r
# Combining vectors into matrix using rbind()
countries <- c('italy','france','germany')
age <- 25:27
rbind(countries,age)
```

```
##           [,1]     [,2]     [,3]
## countries "italy" "france" "germany"
## age       "25"    "26"     "27"
```

```r
# Or using cbind()
cbind(countries,age)
```

```
##      countries age
## [1,] "italy"   "25"
## [2,] "france"  "26"
## [3,] "germany" "27"
```

Arrays in R Programming

Arrays are objects that can have one, two, or more dimensions. When the array is one-dimensional, it coincides with the vector. In the case it's 2D, it's like to use the matrix function. In other words, arrays are useful to build a data structure with more than 2 dimensions.

```r
# Creating an array
a <- array(1:16,dim=c(6,3,2))
a
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    7   13
## [2,]    2    8   14
## [3,]    3    9   15
## [4,]    4   10   16
## [5,]    5   11    1
## [6,]    6   12    2
##
## , , 2
##
##      [,1] [,2] [,3]
```

```
## [1,]    3    9   15
## [2,]    4   10   16
## [3,]    5   11    1
## [4,]    6   12    2
## [5,]    7   13    3
## [6,]    8   14    4
```

list

The list is a ordered collection of objects. For example, it can a collection of vectors, matrices. Differently from vectors, the lists can contain values of different type. They can be build using the function `list()`:

```
# Creating a list
x <- 1:3
y <- c('a','b','c')
l <- list(x,y)
l
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```