manipalglobal
Academy of Data Science

Introduction to Programming

Table of Content

Introduction

This topic emphasises on getting started with programming for a novice audience using the R programming language. On successful completion of this module, the learner would be able to understand the basics of programming in R through a hands-on experience on R data objects.

Learning Objectives

Upon completion of this topic, you will be able to:

- List and explain different data types supported in R
- Define the term data structure and explain the different kinds of data structures
- Perform simple programming operations on data types and data structures

1. Data Types

Data types refer to what kind of data is stored in the variables.

Example: a=10

The variable “a” holds a value 10 and its type is numeric. As the variable “a” can hold only one value at a time, it is called as an atomic type.

The several atomic types supported in R include the following:

Character	Numeric	Integer
Logical	Complex	Date and Time

Variables in R, are also known by the name “objects”. The command used to find the type of a variable is “class()”.

1.1 Character

Character types are used to represent a string value which is enclosed in double or single quotes (“ ” or ‘ ’).

Example 1:

```
s <- “a string”
class(s)
```

Output:

```
[1] “character”
```

In this example, the variable “s” is of the type character.

Example 2:

```
s <- 'abc'
 class(s)
```

Output:

```
[1] "character"
```

Note: The above can also be executed using `s= "a string"` (using the "`=`" operator)

1.2 Numeric

All the numbers are grouped into a single class called "numeric", which includes integer and decimal numbers under its family, basically to represent numbers.

Example:

```
v <- 23.5
class(v)
```

Output:

```
[1] "numeric"
```

In this example, the variable "v" is of the type numeric.

1.3 Integer (Whole numbers: The numbers without a decimal point)

Integers (by default) are of type “numeric” in R, which needs to be converted to integers through some workaround (type casting or type coercion).

To explicitly typecast a variable to an integer type, the following command is used:

```
a=10
as.integer(a)->b
```

The variable “a” of type numeric (by default), is explicitly converted to integer and stored into another variable “b”.

OR

```
as.integer(10)->a
```

The value 10 of type numeric (by default) is converted to an integer and then stored in an object “a”.

OR

```
a=as.integer(20)
```

This can also be achieved using the assignment operator (“=” equal to).

Example:

```
as.integer(20) -> b
class(b)    #This would print that the class of variable b is of type
"integer"
```

Anything which is written after "#" is considered as a comment in R and that part will not be executed.

Example:

```
a=10 #A is of type numeric
```

`#A is of type numeric` - this is a comment and will not be executed.

Example:

```
c = as.double(11.11)
class(c)
```

Output:

```
[1] "numeric"
```

Note: All the float/double variables belong to the category of numeric.

1.4 Logical

Logical type holds the values TRUE/FALSE (TRUE is different from true, case-sensitive)

Example 1:

```
a = TRUE   #A logical value is assigned to "a"
class(a)
```

Output:

```
[1] "logical"
```

The above example shows that “a” is a variable of type Logical.

Note: TRUE/FALSE always has to be in upper case.

Example 2:

```
a = 10
b = 20
c = a > b   # is a greater than b?
print(c) # prints the logical value stored in "c", that is
```

Output:

```
FALSE
```

1.5 Complex

Complex type is used to store complex numbers (mathematics) 3+2i. In R, it is defined through the pure imaginary value “i”.

Example:

```
a = 1 + 5i       # creates a complex number
a                # prints the value of a
class(a)         # print the class name of the variable "a"
```

In this example, “a” is a variable of type complex.

1.6 Date and Time

There are many ways to deal with date and time type in R.

The "as.Date()" function handles dates in R. The POSIXct and POSIXlt classes (calendar time and local time) represent calendar dates and time.

- To get the current date and time, the command would be `date()`
- To know the current date, the command would be `Sys.Date()`

1. Input two dates (initially they are of character types)

```
d1 = "2016-06-27"
d2 = "2017-05-21"
```

2. Convert it into date type

```
as.Date(d1)->newd1
as.Date(d2)->newd2
```

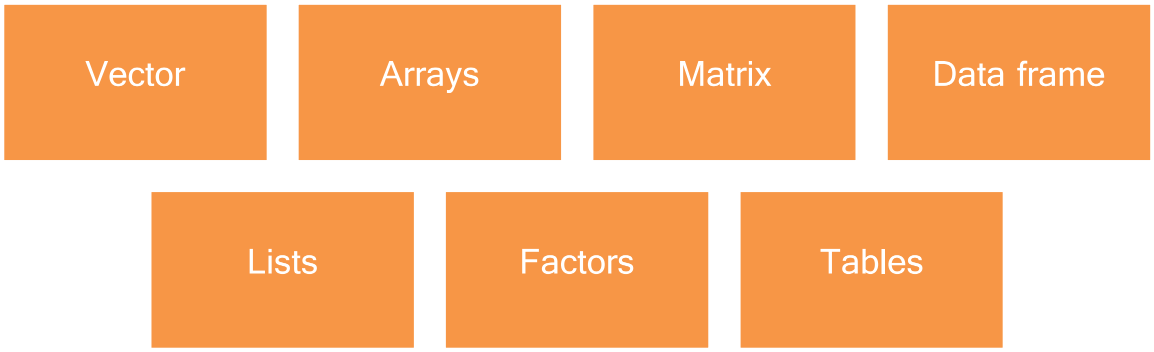
3. Find the difference between the dates

```
newdate <-newd2-newd1
newdate
```

newd2 – newd1 would give the number of days between the two dates.

2. Data Structures

Data structure can be understood as a way of organising and storing the data. In R, there are different kinds of data structures:



The above-mentioned data structures or objects in R, are also called as aggregate data types. They can store more than one value at a time, unlike the atomic types.

2.1 Vectors

A vector is a collection of elements of the same type. The elements of a vector can be logical, integer, character, numeric, and so on.

A vector is created using the **c()** function (concatenate) for grouping the elements.

Example 1:

Numeric Vector:

```
a = c(1,2,3,4)
print(a)
```

Output:

```
[1] 1 2 3 4
```

This example demonstrates how to create a numeric vector and prints the values.

Example 2:

Character Vector:

```
b=c(“a”,“b”,“c”,“d”)
print(b)
```

Output:

```
b
[1] “a” “b” “c” “d”
```

Example 3:

The vector below contains elements of different data types

```
a = c(1, “hello”)
class(a)
```

Output:

```
[1] “character”
```

Typecasting of vector elements takes place automatically. This is known as automatic type coercion in R. Coercion takes place in the order of precedence (in decreasing order with the character being the highest) character, numeric, integer, and logical.

Example 4:

A vector which contains numeric and a logical type inside a vector:

```
a=c(TRUE, 10)
class(a)
```

Output:

```
[1] Numeric
```

Numeric type takes precedence over logical type, that is, TRUE would be converted to its numeric value 1 and the vector would contain (1,10) instead of (TRUE,10).

Note: TRUE is replaced by 1 and the number 10 remains as it is.

A vector containing logical and string types are defined below:

```
x= (TRUE, "a")
```

The resulting vector "x" will now become a character vector, and contents will be "TRUE" and "a". The character type takes precedence over the logical type. Hence, all the vector elements are automatically typecasted to character type.

Working with Vectors:

- Length of a vector:

  ```
  length(a)
  ```

  Length would give the number of elements in a vector.

- Accessing vector locations using index operators:

```
a[2] # This gives the element second position in a vector.
```

- Accessing vector location items within a range:

```
a[2:5] #This gives elements in a range between the second and the fifth
position.
```

- Accessing multiple elements from the vector “a” in multiple locations:

  The vector below contains a set of numeric values:

```
a=c(9,1,2,6,3,8,2,0)
```

- To access the vector elements at positions 1, 2 and 5:

  First create a new vector

```
b=c(1,2,5)
```

  To access the elements of vector “a” in the positions 1,2 and 5, the below command is used:

```
a[b]
```

  a[b] would be equivalent to a[1,2,5]

Note: Directly using the syntax a[1,2,5] will throw an error.

- To find the length of each object in the vector, the `nchar()` function is used:

```
mystr = c("b","bc","acdf")
nchar(mystr)  #This function gives the length of each object
```

2.2 Array

An array is a multidimensional way of storing data in the form of rows and columns. In arrays, the vector elements are arranged into rows and columns as specified by the user.

Example 1:

```
a=1:4
d=c(2,2)
myarr=array(a,d)
```

The above statement will produce a 2 dimensional array with 2 rows and 2 columns.

Example 2:

To create an array with 2 rows and 3 columns run the code below:

```
s=c(1,2,3,4)
d=c(2,3)
myarr = array(s,d)
myarr
```

Output:

```
          [,1]  [,2] [,3]
[1,]       1     3    1
[2,]       2     4    2
```

Note: The first two elements are repeated to complete the array of size 2x3.

Example 3:

```
a=1:4

d=c(2,2) # The vector "d" specifies the number of rows and
#columns that the data needs to be arranged into

myarr=array(a,d)
```

Output:

```
        [,1] [,2]
[1,]     1     3
[2,]     2     4
```

2.3 Matrix

Matrices are a collection of homogenous elements in the form of rows and columns, in a tabular layout. All columns in a matrix should have the same class (numeric or character) and the same length.

Creation of a Matrix

Matrix is created using the **matrix()** function, and its dimensions can be specified using **nrow** and **ncol.** In case, if any one of the dimension is specified, the other dimension is inferred automatically.

Syntax:

```
matrixSample <- matrix(vector, nrow=r, ncol =c, byrow=FALSE,
dimnames=list(char_vector_rownames, char_vector_colnames))
```

`byrow = TRUE,` indicates matrix should be filled by rows

`byrow = FALSE,` indicates matrix should be filled by columns (default)

`dimnames =` specifies labels for rows and columns (optional)

Example 1:

```
mymat = matrix(c(1:10),nrow=5, byrow = TRUE)
print(mymat)
```

Output:

```
          [,1] [,2]
[1,]        1     2
[2,]        3     4
[3,]        5     6
[4,]        7     8
[5,]        9    10
```

Ten elements are arranged into a table format (rows and columns), where the number of rows is mentioned as 5.

Note: In the above example, the byrow value is set to true. Hence, the matrix is constructed where the elements are filled in row wise.

Example 2:

```
mymat = matrix(c(1:10),nrow=5, byrow = FALSE)
print(mymat)
```

Output:

```
          [,1]  [,2]
[1,]      1     6
[2,]      2     7
[3,]      3     8
[4,]      4     9
[5,]      5    10
```

Note: In the above example, the byrow value is set to false. Hence, the matrix is constructed where the elements are filled column wise.

Example 3:

The length of the vector "a" should be a product of the number of rows and columns.

```
a = 1:6
mymat= matrix(a,nrow=2, ncol = 3, byrow = TRUE)
```

Working with Matrix:

- Accessing the matrix elements of 1st row and 1st col of the matrix mymat

```
mymat[1,1] # the element at position 1,1 is retrieved
```

Output:

```
[1] 1
```

- Printing all the elements of 2nd column of the matrix mymat

```
mymat[,2] # all the elements of the second column of the matrix are
retrieved
```

Output:

```
[1] 2 5
```

- Printing all the elements of 1st row of the matrix mymat

```
mymat[1,] # all the elements of the first row of the matrix are retrieved
```

Output:

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

- Checking if elements of 2nd column are greater than a number specified by the user

```
mymat[,2] > 3
```

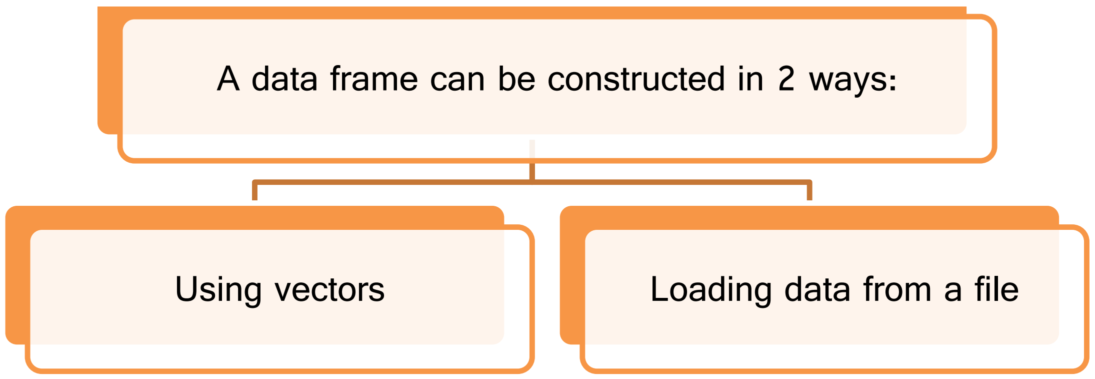
Output:

```
[1] FALSE  TRUE
```

Note: There is no significant difference between 2-dimensional arrays and matrices. However, matrices are a preferred choice among programmers when compared to arrays.

2.4 Data Frames

Data frames are used to represent data in a tabular format, arranged into rows and columns. The elements of a data frame can be of heterogeneous by nature.



Working with Data Frames:

- Creating a data frame using vectors

Example:

The data frame below is created using three vectors a, b and c.

```
a =1:5
b= 6:10
c=11:15
data.frame(a,b,c) -> df
print(df)
```

Output:

```
  a  b  c
1 1  6 11
2 2  7 12
3 3  8 13
4 4  9 14
5 5 10 15
```

Note: All rows must have same number of columns or all vectors must be of same length.

- Accessing the data frame elements
  The "df" data frame is composed of 3 vectors (a, b and c) which is stored as three columns in a data frame. The column names of a data frame would be the same names as that of a vector.

```
print(df$a)    # This command would print the first column from the data
frame
```

Output: `[1] 1 2 3 4 5`

- Command to print the first 2 columns from the data frame

```
a=df[,c(1,2)]
```

Output:

```
    a  b
1 1   6
2 2   7
3 3   8
4 4   9
5 5  10
```

- Command to extract the first row from the data frame

```
row1 = df[1,]
print(row1)
```

Output:

```
  a b  c
1 1 6 11
```

- Command to extract the first 2 rows from the data frame

```
row1to2 = df[1:2,]
print(row1to2)
```

Output:

```
    a b  c
1 1 6 11
2 2 7 12
```

- To access all the elements of row 1 and 5

```
row1and5 = df[c(1,5),]
```

Output:

```
   a  b  c
1 1  6 11
5 5 10 15
```

- To access only the 3rd column elements of row 1 and 5

```
row15andcol3 = df[c(1,5),3]
```

Output:

```
[1] 11 15
```

Usage of Head () function

If a data frame has 20 rows, the first 6 rows of the data frame can be accessed using the following command:

```
head(name of the data frame)
```

By default, the head command will display only the first 6 rows in a data frame.

- Head()

```
Head(df, 20) # this command will retrieve the first 20 rows
#of the data frame, where df is the name of the data frame.
```

- Tail()

  Last 6 rows of a data frame are retrieved using the tail command

  ```
  tail(df)
  ```

  Tail function works exactly same as the head function, but in a bottom-up fashion.

2.5 Lists

List is an ordered collection of objects which contains objects of different types. Usually, a list is constructed using vectors of the same or different type.

Example:

The list below is created using three vectors v1, v2 and v3

```
v1 =c("true","false");
v2= 1:5
v3 = c("cat","dog")
my_list=list(v1,v2,v3) #my_list object is a collection of 3 different
vectors.
```

Output:

```
[[1]]
[1] "true"  "false"

[[2]]
[1] 1 2 3 4 5

[[3]]
[1] "cat" "dog"
```

`my_list` is a collection of type list and it can hold any kind of simple or complex types.

Working with Lists:

- Accessing the elements of the list

```
my_list[1] # First element of the list is retrieved
```

Output:

```
[[1]]
[1] "true"  "false"
```

- To access the first item in the 3rd element of list

```
my_list[[3]][1]
```

Output:

```
[1] "cat"
```

- To command that gives the length of the list

```
length(my_list)
```

2.6 Factors

Factors are objects which are used to store and categorise data. Each value in a factor is stored only once and in the form of levels.

Example 1:

The vector below represents a family of two girls (1) and four boys (0) in the form of factors.

```
kids = factor(c(1,0,1,0,0,0), levels = c(0, 1),labels = c("boy",
"girl"))
kids
class(kids)
```

Output:

```
[1] girl boy girl boy boy boy
Levels: boy girl
 [1] "factor"
```

Example 2:

```
f1 = factor( c ("Pig", "Hive", "Hbase", "Pig"))
nlevels(f1)
```

In this example, `nlevels()` will print the number of unique data in the above list, which is 3.

Output:

```
[1] 3
```

Example 3:

```
f1=factor(c("Hello", "word",11))
f1
```

Output:

```
[1] Hello word  11
Levels: 11 Hello word
```

Note: Factors are also used in plotting graphs while working with categorical data.

Example 4:

```
sizes = c("small", "medium", "large", "small")
f = factor(sizes)
f
```

Output:

```
[1] small medium large small
Levels: large medium small
```

2.7 Tables

Table() function is used to create tabular results of categorical variables. It is used for generating a frequency distribution of the input data.

Example:

The vector which is created below is given as an input to the `table()` command.

```
a=c(1,1,3,4,5,5,5,3,3,2,10)
table(a)
```

Output:

```
 1  2  3  4  5 10
2  1  3  1  3  1
```

The Table() function gives the frequency count of how many times each number is repeated in the above vector "a".

Working with Tables:

- Table command on a built-in dataframe USArrests

```
table(USArrests[,3])
```

  It will return the frequency count of all the elements in 3rd column of USArrests data frame.

- Factor values are captured in a table format using table()

Example:

```
y <- factor(c("a","b","a","a","b"))
z <- table(y)
z
y
```

Output:

```
y
a b
3 2
```

- Creating a table from a matrix

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

The above command creates a matrix of three rows with elements ranging from 1 to 9.

```
rownames(mymat) = c("r1","r2","r3")
colnames(mymat) = c("c1","c2","c3")
```

rownames and colnames assign names to the elements of the matrix, row and column wise.

```
mytab = as.table(mymat)
print(mytab)
```

Output:

```
     c1 c2 c3
r1   1  2  3
r2   4  5  6
r3   7  8  9
```

The as.table() function creates the matrix in a table format.

- Table() function determines a logical condition.

Example:

The vector x contains a list of numeric values

```
    x <- c (5,12,13,3,4,5)
   table(x == 5)
```

A logical expression is given as an input to the table() command.

Output:

```
   FALSE    TRUE
     4        2
```

The output infers how many number of elements in the vector x are equal to the number 5.

Summary

Data types refer to what kind of data is stored in the variables. The several atomic types supported in R are:

- Character
- Numeric
- Integer
- Logical
- Complex
- Date and Time

Data structure can be defined as a specific form of organising and storing the data. The different kinds of data structures in R are:

- Vector
- Arrays
- Matrix
- Data frame
- Lists
- Factors
- Tables