R language provides programmers with the facility to put elements of different types into a single container like numbers, strings, vectors etc. This flexibility of containing multiple types of elements is possible is R programming using Lists. In this chapter you will learn about the lists and its implementation within R program.

**What are Lists in R programming?**

A list, in lame word, a vector in which all the element can be of a diverse type. It can be handled by first creating lists then using index programmers can manipulate lists.

**Creating Lists**

Lists can be created using the list function and denoting the contents works a lot like the ‘c’ function which you have seen already in previous tutorials. You just list the contents, with each argument and each separated by a comma. List elements can be of any variable type—vectors, numbers or even functions. Here is a simple example of how to use list in R:

alist <- list ("Red", "Blue", c(42,36,01), FALSE, 73.91, 128.6)

print (alist)

Here, alist is the name of the list, list() is use to lists all the elements of different types. And the next is the print statement which prints the entire variable’s value.

**Atomic and Recursive Variables**

Due to this capability of containing other lists within themselves (lists), lists are considered as the recursive variables. Vectors and arrays, by default, come under the category of atomic variables. Variables can either be recursive or atomic but never both. The functions is.recursive and is.atomic let programmers’ test variables to see what their types are. Here is a simple example of using these 2 functions –

is.atomic (list ())

# [1] FALSE                  the output comes as False

is.recursive (list ())

# [1] TRUE                   the output comes as True

is.atomic (numeric ())

# [1] TRUE                   the output comes as True

**Indexing Lists in R Language**

Consider this test list:

ls <- list(

first = 2,

second = 4,

third = list(

fourth = 3.2,

fifth = 6.4

)

)

As with vectors, you can access every  element of the list by the use of square brackets – [], and by numeric indices or using element names or by the logical index. Here is an example of how o use these values

ls [1:2]

# $first

# [1] 2

# $second

# [1] 4

ls[-3]

# $ first

# [1] 2

# $ second

# [1] 4

Or the lists values can also be invoked using element’s name as:

ls [c ("first", "second")]

# $ first

# [1] 2

# $ second

# [1] 4

**Manipulating List Elements**

You can insert, delete, update and modify list elements within a list. You can insert and delete elements just at the end of any list. But list allows to update any element.

names (alist) <- c ("Ray", "Karl", "Steve")

# Now you will add element at the end of the list

alist[4] <- "Mark"

print (alist[4])

# Now you will remove the last element from the list

alist[4] <- NULL

# Now you will update the 2nd Element

alist [2] <- "Karlos"

print (alist[2])

**List**

A list is a generic vector containing other objects.

For example, the following variable x is a list containing copies of three vectors n, s, b, and a numeric value 3.

> n = c(2, 3, 5)

> s = c("aa", "bb", "cc", "dd", "ee")

> b = c(TRUE, FALSE, TRUE, FALSE, FALSE)

> x = list(n, s, b, 3) # x contains copies of n, s, b

**List Slicing**

We retrieve a list slice with the single square bracket "[]" operator. The following is a slice containing the second member of x, which is a copy of s.

> x[2]

[[1]]

[1] "aa" "bb" "cc" "dd" "ee"

With an index vector, we can retrieve a slice with multiple members. Here a slice containing the second and fourth members of x.

> x[c(2, 4)]

[[1]]

[1] "aa" "bb" "cc" "dd" "ee"

[[2]]

[1] 3

**Member Reference**

In order to reference a list member directly, we have to use the double square bracket "[[]]" operator. The following object x[[2]] is the second member of x. In other words, x[[2]] is a copy of s, but is not a slice containing s or its copy.

> x[[2]]

[1] "aa" "bb" "cc" "dd" "ee"

We can modify its content directly.

> x[[2]][1] = "ta"

> x[[2]]

[1] "ta" "bb" "cc" "dd" "ee"

> s

[1] "aa" "bb" "cc" "dd" "ee" # s is unaffected

**Named List Members**

We can assign names to list members, and reference them by names instead of numeric indexes.

For example, in the following, v is a list of two members, named "bob" and "john".

> v = list(bon=c(2, 3, 5), john=c("aa", "bb"))

> v

$bon

[1] 2 3 5

$john

[1] "aa" "bb"

List Slicing

We retrieve a list slice with the single square bracket "[]" operator. Here is a list slice containing a member of v named "bob".

> v["bon"]

$bon

[1] 2 3 5

With an index vector, we can retrieve a slice with multiple members. Here is a list slice with both members of v. Notice how they are reversed from their original positions in v.

> v[c("john", "bon")]

$john

[1] "aa" "bb"

$bob

[1] 2 3 5

Member Reference

In order to reference a list member directly, we have to use the double square bracket "[[]]" operator. The following references a member of v by name.

> v[["bon"]]

[1] 2 3 5

A named list member can also be referenced directly with the "$" operator in lieu of the double square bracket operator.

> v$bon

[1] 2 3 5

Search Path Attachment

We can attach a list to the R search path and access its members without explicitly mentioning the list. It should to be detached for cleanup.

> attach(v)

> bon

[1] 2 3 5

> detach(v)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. R List

In Our previous R tutorial, we have discussed what is R Data Types in detail. Now in this R programming tutorial, we will see one of the R data Types i.e. R List in detail. First of all, we will learn what is R List, then we will also discuss how to Create list in R, how to Access R list, how to modify lists in R with the help of examples.

Introduction to R List and its operations with examples.

Introduction to R list

2. What is R List?

R List is the object which Contains elements of different types – like strings, numbers, vectors and another list inside it. R list can also contain a matrix or a function as its elements. The List is been created using list() Function in R. In other words, a list is a generic vector containing other objects.

For Example:

The variable x is containing copies of three vectors n, s, b and a numeric value 3.

n = c(2, 3, 5)

s = c(“aa”, “bb”, “cc”, “dd”, “ee” )

b = c(TRUE, FALSE, TRUE, FALSE, FALSE )

x = list( n, s, b, 3) # x contains copies of n, s, b

3. How to Create Lists in R Programming?

In this section of R List tutorial, we will create R List with the help of an example.

Let’s create a list containing string, numbers, vectors and logical values.

For Example:

List\_data <- list("Green", "Yellow", c(5,6,7), TRUE, 51.2)

print(list\_data)

When we execute the above code, it produces the following result-

[[1]]

[1] “Green”

[[2]]

[1] “Yellow”

[[3]]

[1] 5, 6, 7

[[4]]

[1] TRUE

[[5]]

[1] 51.2

4. How to Name List Elements in R Language?

In this section, we will discuss to name the R List elements with the help of an example.

Let’s create a list containing a vector, a matrix, and a list.

For Example:

list\_data <- list(c("Feb","Mar","Apr"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))

Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

Show the list.

print(list\_data)

When we execute the above code, it produces the following

Result-

$1<sup>st</sup> Quarter’[1] “Feb”, “Mar”, “Apr”

$A Matrix

[,1] [,2] [,3]

[1,] 3 5 -2

[2,] 9 1 8

$A\_Inner\_list

$A\_Inner\_list [[1]]

[1] “Green”

$A\_Inner\_list [[2]]

[1] “12.3”

5. How to Access R List Elements?

Let’s now understand how to access Lists elements in R programming?

Create an R list containing a vector, a list and a matrix. list\_data <- list(c(“Feb”,”Mar”,”Apr)) list(“white”,13.4)), matrix(c(3,9,5,1,-2,8), nrow = 2)

For Example:

Give names to the elements in the list.

Names(list\_data) <- c(“1<sup>st </sup>Quarter”, “A Matrix”, “A Inner list”)

Access the first element of the list.

print(list\_data[1])

Access the third element. As it also a list, all its elements will print.

Print(list\_data[3])

By using the name of the element access the list elements.

Print(list\_data$A Matrix)

It will produce the following result after executing the above code-

$”1st Quarter” [1] “Feb”, Mar”, Apr”

$A\_Inner\_list

$A\_Inner\_list[[1]]

[1] “White”

$A\_Inner\_list[[2]]

[1] 13.4

$ “A Matrix” [1]

[1] [2] [3]

[1] 3 5 -2

[2] 9 1 8

6. How to Manipulate List elements in R Programming?

Let’s now discuss how to manipulate the R List elements with the help of an example.

Create a list containing a vector, a matrix and a list.list\_data <- list(c(“Feb”,”Mar”,”Apr”), matrix(c(3,9,5,1,-2,8), nrow = 2),list(“green”,12.3))

For Example:

Give names to the elements in the list.

names(list\_data) <- c("1st Quarter", "A\_Matrix", "A Inner list")

Add an element at the end of the list.

list\_data[4] <- "New element"print(list\_data[4])

Remove the last element.

list\_data[4] <- NULL # Print the 4th Element.print(list\_data[4])

Update the 3rd Element.

list\_data[3] <- "updated element"print(list\_data[3])

When we execute the above code, it produces the following result.

[[1]]

[1] "New element"

$NULL

$`A Inner list`

[1] "updated element"

7. How to Merge Lists in R Programming language?

We can merge many lists into one list by placing all the lists element inside one list() function.

R Interview Questions and Answers

For Example:

Create two lists.

list1 <- list(1,2,3)list2 <- list("Mon","Tue","Wed")

Merge the two lists.

list <- c(list1,list2) # Print the merged list.print(merged.list)

When we execute the above code, it produces the following result.

[[1]]

[1] 1

[[2]]

[1] 2

[[3]]

[1] 3

[[4]]

[1] "Mon"

[[5]]

[1] "Tue"

[[6]]

[1] "Wed"

8. How to Convert R List to Vector?

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can apply after the list is converted into vector. To do this conversion, we can use the unlist() function. It takes the list as input and produces a vector.

For example:

Create lists.

list1 <- list(2:6)

print(list1)

list2 <-list(11:15)

print(list2)

Convert the lists to vectors.

v1 <- unlist(list1)

v2 <- unlist(list2)

print(v1)

print(v2)

Now add the vectors

result <- v1+v2

print(result)

When we execute the above code, it produces the following result −

[[1]]

[1] 2 3 4 5 6

[[1]]

[1]11 12 13 14 15

[1] 2 3 4 5 6

[1]11 12 13 14 15

[1] 13 15 17 19 21

9. How to Generate Lists in R?

We can use a colon to generate a list of numbers.

For example:

-3:3

[1] -3 -2 -1 0 1 2 3

10. Operating on lists in R

R allows to Operate on all list values at once.

For example:

c(1,3,5) + 4

This and the Apply function allow you to avoid most for loops.

[1] 5, 7, 9

11. R Predefined Lists

Lists for letters and month names are predefined:

letters

[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o"

[16] "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

LETTERS

[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O"

[16] "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"

month.abb

[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct"

[11] "Nov" "Dec"

month.name

[1] "January" "February" "March" "April" "May"

[6] "June" "July" "August" "September" "October"

[11] "November" "December"

12. The c function

The c Function combines the parameter into a list and converts them to the same type.

For example:

c("test",4)

[1] "test" "4"

typeof("4")

[1] "character"

Here 4 is converts into a string.

This was all in R list tutorial.

13. Conclusion

Hence, we have studied about R list in above-mentioned information. We all are aware, that lists are the object which contains elements of different types like strings, numbers, and vectors. Thus it is necessary to learn how to apply different operations on list elements.