STATISTICAL PROGRAMMING IN R

K G Srinivasa
Associate Professor & Head
Department of Information Technology
Ch. B.P. Government Engineering College
Jaffarpur, Delhi

G M Siddesh
Associate Professor
Department of Information Science & Engineering
Ramaiah Institute of Technology
Bengaluru

Chetan Shetty
Assistant Professor
Department of Computer Science & Engineering
Ramaiah Institute of Technology
Bengaluru

B J Sowmya
Assistant Professor
Department of Computer Science & Engineering
Ramaiah Institute of Technology
Bengaluru

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trademark of
Oxford University Press in the UK and in certain other countries.

Published in India by
Oxford University Press
Ground Floor, 2/11, Ansari Road, Daryaganj, New Delhi 110002, India

First published in 2017

ISBN-13: 978-0-19-948035-7
ISBN-10: 0-19-948035-4

Typeset in Times New RomanPSMT
by The Composers, New Delhi 110063
Printed in India by Magic International (P) Ltd., Greater Noida

Preface

R is an open source, interpreted programming language and interactive development environment for high performance statistical computing and effective data visualization. It is similar to other statistical packages like the S language that was originally developed by Bell Labs, USA. Nowadays, it is a widely accepted open source solution for high dimensional data analytics supported by a dynamic and vibrant research community. A majority of the data analysts and data scientists around the globe utilize R programming to tackle challenging issues in fields ranging from computational science to business marketing. R programming has become the most common programming language of the data science community and for various finance- and analytics-driven business organizations such as Google, Facebook, and LinkedIn.

R and its libraries include support for different statistical and graphics related functions along with linear and non-linear modelling, time-sequence analysis, and data mining techniques such as clustering and classification. R can also be used as an extension service with other packages like Hadoop. Further, the open source community has developed a number of plug-ins and extensions to a variety of applications ranging from health care to business intelligence. R allows application developers to select the algorithms of their choice and develop packages of their own. For computationally intensive tasks, other programming language (e.g., C, C++, and Python) codes can be linked with R in run-time. Users can write C, C++, Java, .NET, or Python code to control R objects. Further, R has more than 5000 packages including libraries and functions that support various focused applications such as cosmology, physical sciences, genomics, drug advancement, finance, health care, advertisement, and many others. Hence, it is very easy for the application developers to start building applications using R.

ABOUT THE BOOK

Statistical Programming in R is a textbook designed for the first course on the subject taught for the students of undergraduate engineering in computer science and computer applications. The book would also be useful for people who are beginners in data science and statistical analysis, and those who want to begin with a hands-on approach using R.

While skipping any chapter is not recommended, the book is organized in such a way that the readers can refer to any chapter of their choice to get an overall grasp of the related concepts. *Chapters 1–3* discuss the basic data structures used in R. *Chapters 4–6* cover the basic programming constructs such as conditional, relational, and logical operators as well as loops and functions. *Chapter 7* introduces the apply family of functions that are widely used in R. *Chapter 8* details plotting in R and *chapter 9* covers the reading and writing into various types of data interfaces. *Chapter 10* combines concepts from all the previous chapters and provides a comprehensive set of examples that clearly show how data analysis is performed in R.

KEY FEATURES

- Provides an in-depth introduction to R that is easy to follow for both beginners in statistical analysis and data science as well as experts who want to get started with R.
- Addresses all topics required to perform analysis on real data, ranging from reading data stored in various file formats to plotting the results of the analysis.
- Contains numerous examples such as binary search tree implementation, accessing keyboard and monitor for general input and output, and many others
- Explains the various constructs in R and the nuances between them

- Illustrates how R can interface with CSV, Excel, XML, JSON files
- Provides lucid examples covering ANOVA, advanced statistics, splines, and also covers data visualization through R

CONTENTS AND COVERAGE

The chapter wise scheme of the book is given here.

Chapter 1 deals with the basics of R. It provides the readers everything they need to know to get started with R from the programming environments to basic data structures such as matrices, vectors, arrays, and classes.

Chapter 2 discusses factors and data frames. It provides users with the understanding they require to use these data structures including the concepts of creation, comparison, and subsetting.

Chapter 3 introduces lists. Through this chapter, the reader will gain an understanding on what lists are and how to use them effectively. It also covers basic operations on lists such as accessing, manipulation, merging, as well as conversion of lists to vectors.

Chapter 4 deals with conditional operators in R. This includes relational, logical, and conditional operators, their use on vectors and the nuances between the logical operators.

Chapter 5 introduces loops in R. It covers the use of for and while loops in R and their use on the data structures of R.

Chapter 6 covers functions in R, beginning with creation of user-defined functions, scope, loading packages in R, various math functions, and discusses various calculus functions and their advanced operations. This chapter includes examples for binary search tree implementation and accessing keyboard and monitor for general input and output.

Chapter 7 introduces the family of apply functions in R. It covers the usage of these functions and the differences between them.

Chapter 8 deals with the various mechanisms of plotting in R. It covers in great depth the many types of plots available in R, the differences between them, and how they can be made.

Chapter 9 discusses the various data interfaces in R. It covers the different types of data interfaces and how files can be read and written into these interfaces.

Chapter 10 provides a diverse set of examples combining concepts from Chapters 1–9 to give a taste of the kind of analysis that R is used for. It covers various regression mechanisms and statistical hypothesis tests.

ONLINE RESOURCES

The following resources are available to support the faculty and students using this text.

For faculty

- Solution manual: Complete solutions to select exercise problems from each chapter of the book
- PowerPoint lecture slides: PowerPoint lecture material in bulleted lists modularized according to chapters

For students

- Useful web links

ACKNOWLEDGEMENTS

We attribute our efforts for completing this book to all the people who have inspired us and shaped our careers. We thank our college management, colleagues, and students who encouraged us to work on this book.

Srinivasa thanks Prof. Vijay K. Minocha, Principal of CBP Government Engineering College, New Delhi, who encouraged him in working on this book. His special thanks to his esteemed colleagues Prof. Harne, Dr Athar Hussain, Pankaj Lathar, and Seema Rani for their valuable suggestions and continuous encouragement.

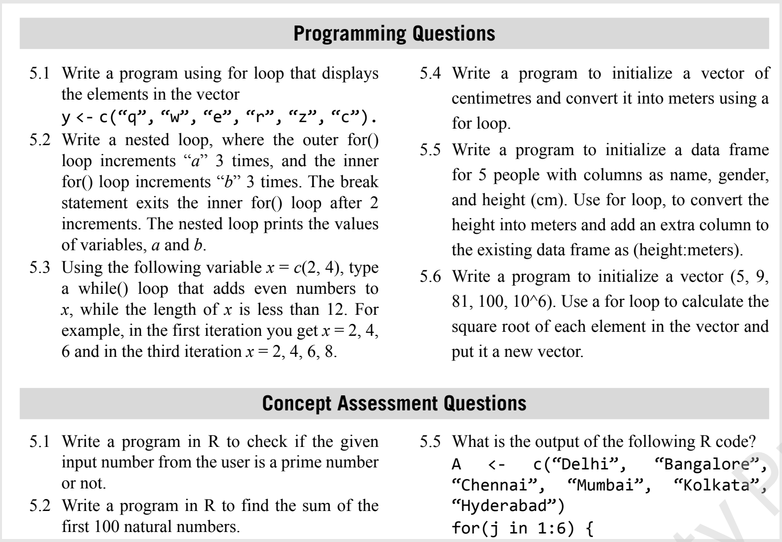
Both Siddesh and Srinivasa thank Dr N.V.R. Naidu, Principal, Ramaiah Institute of Technology for his valuable guidance and support. Special thanks to Dr T.V. Suresh Kumar, Dr B.P. Vijay Kumar, and Mr Ramesh Naik for their continuous encouragement. The authors acknowledge the valuable inputs and suggestions given by Dr Anita K., Dr Seema S., Mr Srinidhi H., Mr Manishekar, and Mr Sandeep Bhat.

We are extremely grateful to our families, who graciously accepted our inability to attend to family chores during the course of writing this book, and their extended warmth and encouragement. Without their support, we would not have been able to complete this venture.

Last, but not the least, we express our heartfelt thanks to the editorial team at the Oxford University Press, India who guided us through this project.

K.G. Srinivasa
G.M. Siddesh
Chetan Shetty
Sowmya B.J.

Features of



Programming Questions

5.1 Write a program using for loop that displays the elements in the vector
`y <- c("q", "w", "e", "r", "z", "c").`

5.2 Write a nested loop, where the outer for() loop increments "*a*" 3 times, and the inner for() loop increments "*b*" 3 times. The break statement exits the inner for() loop after 2 increments. The nested loop prints the values of variables, *a* and *b*.

5.3 Using the following variable $x = c(2, 4)$, type a while() loop that adds even numbers to *x*, while the length of *x* is less than 12. For example, in the first iteration you get $x = 2, 4, 6$ and in the third iteration $x = 2, 4, 6, 8$.

5.4 Write a program to initialize a vector of centimetres and convert it into meters using a for loop.

5.5 Write a program to initialize a data frame for 5 people with columns as name, gender, and height (cm). Use for loop, to convert the height into meters and add an extra column to the existing data frame as (height:meters).

5.6 Write a program to initialize a vector (5, 9, 81, 100, 10^6). Use a for loop to calculate the square root of each element in the vector and put it a new vector.

Concept Assessment Questions

5.1 Write a program in R to check if the given input number from the user is a prime number or not.

5.2 Write a program in R to find the sum of the first 100 natural numbers.

5.5 What is the output of the following R code?

```
A  <-  c("Delhi",  "Bangalore",
"Chennai",  "Mumbai",  "Kolkata",
"Hyderabad")
for(j in 1:6) {
```

Loaded with learning features
Presents multiple-choice questions, programming questions, and concept assessment questions at the end of all relevant chapters.

Data visualization concepts
Illustrates data visualization through plotting techniques such as bar charts and pie charts.

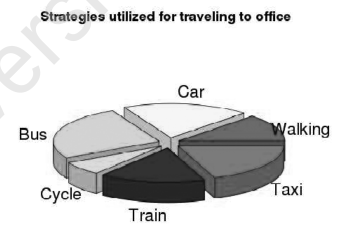
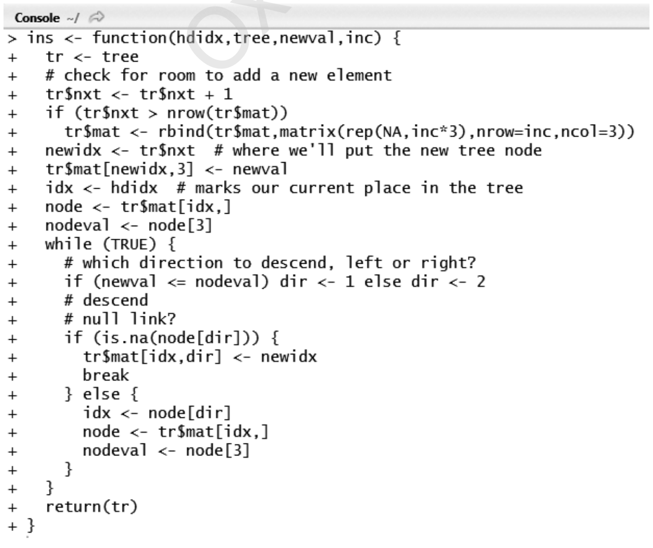


Fig. 8.10 Commands and output for 3D pie chart



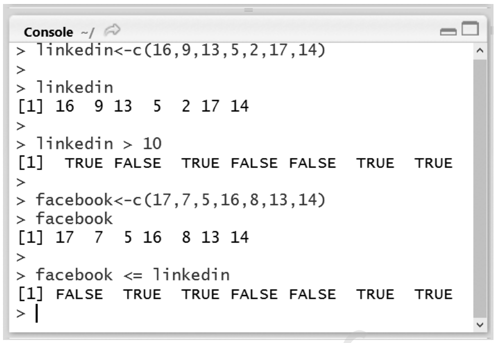
```
> ins <- function(hdidx,tree,newval,inc) {
+   tr <- tree
+   # check for room to add a new element
+   tr$nxt <- tr$nxt + 1
+   if (tr$nxt > nrow(tr$mat))
+     tr$mat <- rbind(tr$mat,matrix(rep(NA,inc*3),nrow=inc,ncol=3))
+   newidx <- tr$nxt  # where we'll put the new tree node
+   tr$mat[newidx,3] <- newval
+   idx <- hdidx  # marks our current place in the tree
+   node <- tr$mat[idx,]
+   nodeval <- node[3]
+   while (TRUE) {
+     # which direction to descend, left or right?
+     if (newval <= nodeval) dir <- 1 else dir <- 2
+     # descend
+     # null link?
+     if (is.na(node[dir])) {
+       tr$mat[idx,dir] <- newidx
+       break
+     } else {
+       idx <- node[dir]
+       node <- tr$mat[idx,]
+       nodeval <- node[3]
+     }
+   }
+   return(tr)
+ }
```

Fig. 6.13 Creating a binary search tree

Interesting examples
Presents multiple examples such as binary search tree implementation and accessing keyboard and monitor for general input and output.

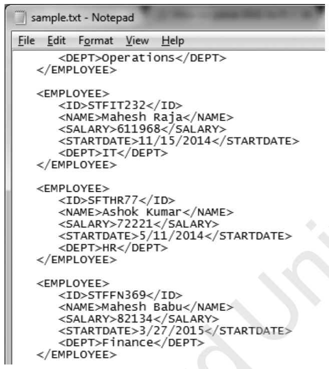
the Book

Constructs and their functionality
Explains various constructs in R such as conditional, relational, and logical operators, their relationship with vectors, conditional statements, and their purpose.



```
> linkedin<-c(16,9,13,5,2,17,14)
>
> linkedin
[1] 16  9 13  5  2 17 14
>
> linkedin > 10
[1]  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE
>
> facebook<-c(17,7,5,16,8,13,14)
> facebook
[1] 17  7  5 16  8 13 14
>
> facebook <= linkedin
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE  TRUE
> |
```

Fig. 4.5 Relational operators and vectors



```
      <DEPT>Operations</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>STFIT232</ID>
      <NAME>Mahesh Raja</NAME>
      <SALARY>611968</SALARY>
      <STARTDATE>11/15/2014</STARTDATE>
      <DEPT>IT</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>SFTHR77</ID>
      <NAME>Ashok Kumar</NAME>
      <SALARY>72221</SALARY>
      <STARTDATE>5/11/2014</STARTDATE>
      <DEPT>HR</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <ID>STFFN369</ID>
      <NAME>Mahesh Babu</NAME>
      <SALARY>82134</SALARY>
      <STARTDATE>3/27/2015</STARTDATE>
      <DEPT>Finance</DEPT>
   </EMPLOYEE>
```

Fig. 9.9 Contents of xml file in notepad

Interfacing with various file formats
Explains how to use R for data available in various file formats such as CSV, Excel, XML, and JSON.

10.15 ANOVA

ANOVA short for analysis of variance, is an analytics method used in statistics. The function of ANOVA is to evaluate the means of the response variable at different factor levels; here the continuous response variable is used with one categorical factor and at least two or more levels.

ANOVA is a function for evaluating hypothesis, to show that there is no difference between at least two means, used when there are no other statistical tools for working. The aov() function is used in R programming to perform the difference in means. The following are the types of ANOVAs:

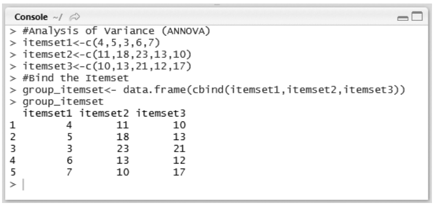
One way ANOVA Contains one fixed factor and only one variable.

Balanced ANOVA Contains any number of factors either fixed or random.

General linear model Allows the user to create covariate and unbalanced designs.

Let us take itemset1—a normal group, itemset2—an announced layoffs, and itemset3—during layoffs to measure stress levels after layoffs. First set up the commands to load the data into tables or vectors and these are the actual R commands to set up the data initially. The data needs to be combined into a single group as shown in Fig. 10.47(a).

Dedicated chapter on simple statistical applications
Provides lucid examples covering ANOVA and advanced statistics.



```
> #Analysis of Variance (ANNOVA)
> itemset1<-c(4,5,3,6,7)
> itemset2<-c(11,18,23,13,10)
> itemset3<-c(10,13,21,12,17)
> #Bind the Itemset
> group_itemset<- data.frame(cbind(itemset1,itemset2,itemset3))
> group_itemset
  itemset1 itemset2 itemset3
1        4       11       10
2        5       18       13
3        3       23       21
4        6       13       12
5        7       10       17
> |
```

Fig. 10.47(a) Introduction to ANOVA

Brief Contents

Detailed Contents

Lists

3

Learning Objectives

A systematic study of this chapter will help the reader understand the following concepts:

- Creating a list
- Accessing and manipulating list elements
- Merging lists
- Converting lists to vectors

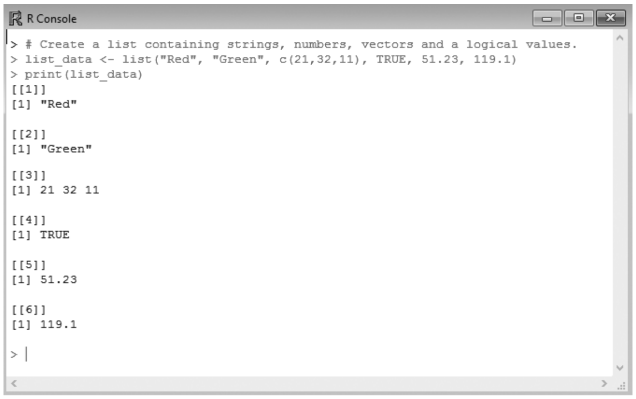
3.1 INTRODUCTION

A list in R constitutes of different objects such as strings, numbers, and vectors. Further, it can include another list within it. Lists in R can also include matrices and functions making it functionally richer than other existing list utilities. Lists can be obtained as return values from many modelling functions such as t.test() for t tests , lm() for liner models, etc. Customized lists can be created to store useful information such as grocery list, shopping list, list of books, and so on. Most of these lists are simple with nominal and ordinal values, but in many cases there is a need for storing complex information such as the configuration details of a phone; in such cases, the list function of R can be used.

3.2 CREATING A LIST

A list can be created using the *list*() function, which takes in different R objects and stores the values in the database.

Figure 3.1 demonstrates the creation of an anonymous list *list_data*, using the *list*() function, containing strings, numbers, vectors, and logical values as data objects. After creating the list it can be viewed with the help of the *print* statement.



```
> # Create a list containing strings, numbers, vectors and a logical values.
> list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
> print(list_data)
[[1]]
[1] "Red"

[[2]]
[1] "Green"

[[3]]
[1] 21 32 11

[[4]]
[1] TRUE

[[5]]
[1] 51.23

[[6]]
[1] 119.1

> |
```

Fig. 3.1 Creation of lists

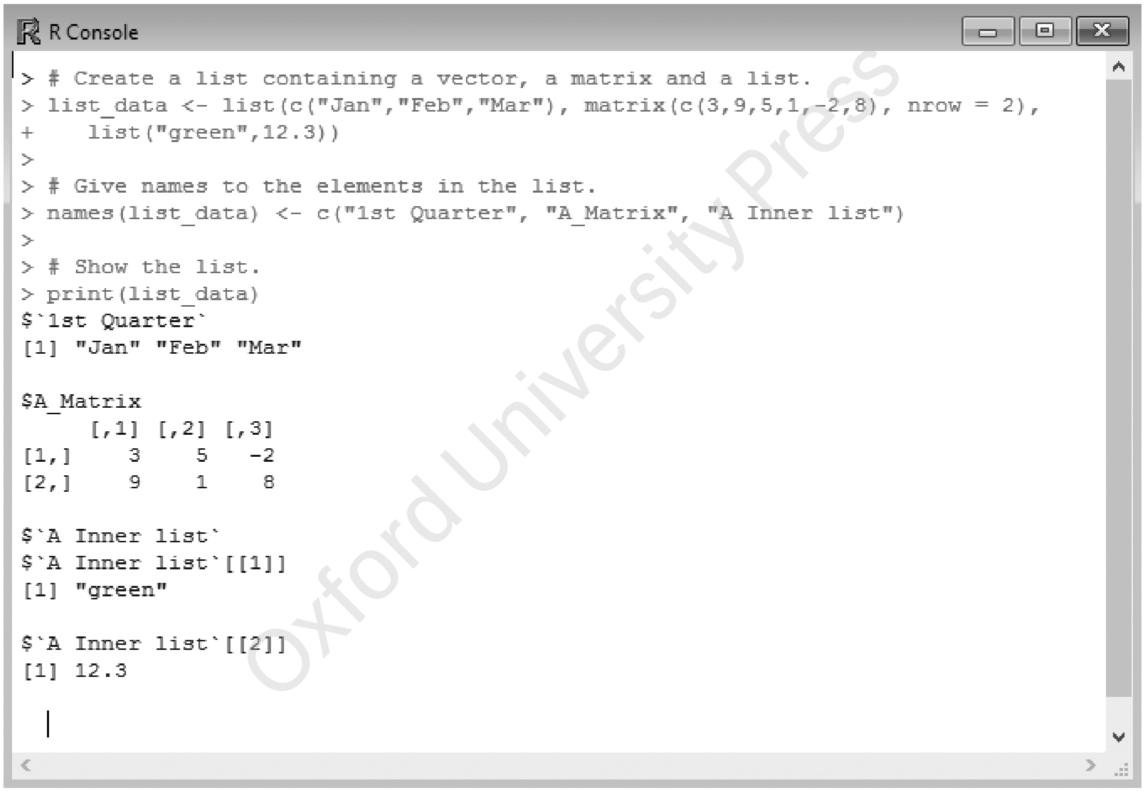
3.2.1 Creating a Named List

Lists in R have a special property wherein names can be assigned to the previously defined set of elements in the list. These forms of named lists provide correct information about a particular category of data with many different attributes. Named lists can be created using the *names*() function.

Example 3.1 Write the command in R console to create a list containing a vector, a matrix, and a list. Also give names to the elements in the list and display the list.

Solution:

Figure 3.2 creates a named list using the *names*() function. First a list called *list_data* containing a vector and a matrix and another list called *list* are created. Later using the *names*() function, a vector with the names to these lists is created. The created list is viewed using the *print* statement.



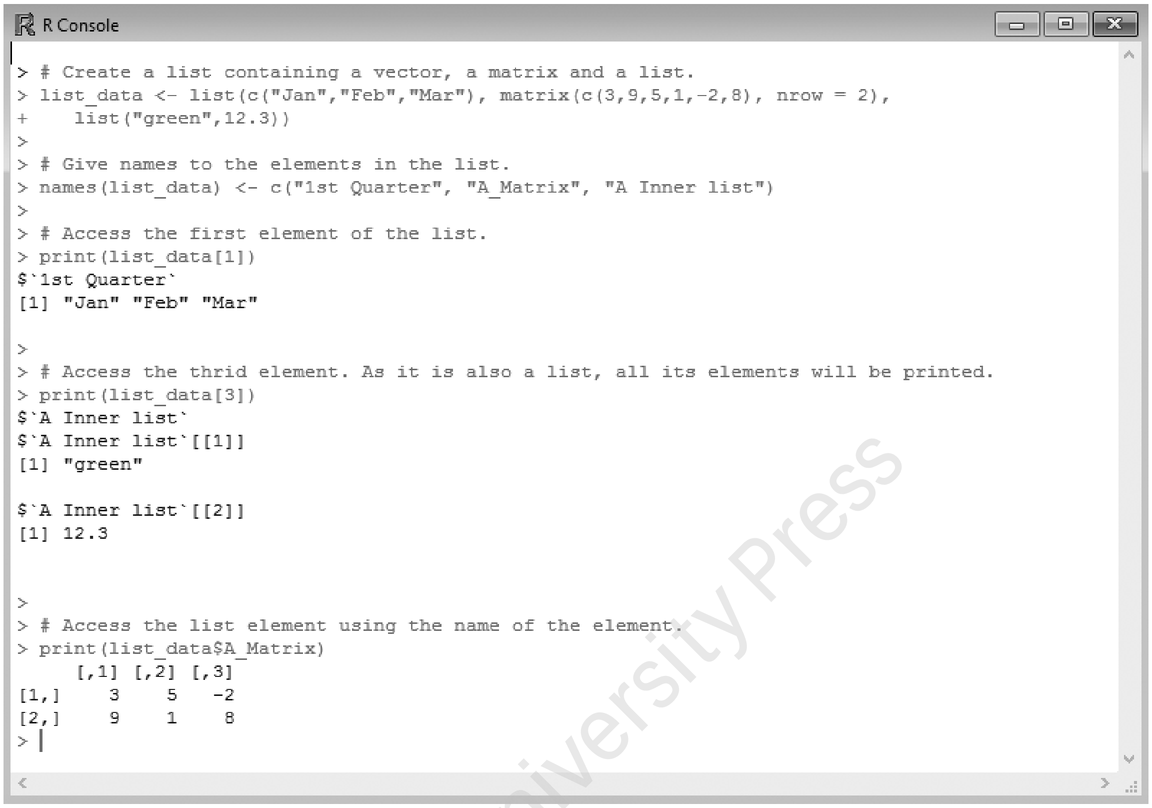
```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Show the list.
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$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
```

Fig. 3.2 Creating a named list

3.3 ACCESSING LIST ELEMENTS

The elements present in a normal list can be accessed using the index value of that element. For a named list, the elements are accessed based on their names.

In Fig. 3.3, a list called *list_data* is created using the *list*() function and later the list elements are given names using the *names*() function. So to access the first element from list_data, the index value of the list is accessed; here the list elements are indexed starting from 1 and not from 0. Therefore printing *list_data*[1] gives the names of the first and later elements in the list. If the third element in the list is considered, which is another list containing two different elements, printing *list_data*[3] gives the list name and the two elements. The elements in the list can also be accessed with their names using the '$' sign appended with the name of



```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), 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")
>
> # Access the first element of the list.
> print(list_data[1])
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

>
> # Access the thrid element. As it is also a list, all its elements will be printed.
> print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

$`A Inner list`[[2]]
[1] 12.3


>
> # Access the list element using the name of the element.
> print(list_data$A_Matrix)
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8
> |
```

Fig. 3.3 Accessing elements from a list

the element in the list. Hence printing *list_data$A_Matrix* gives the output of the third element of the list which is a matrix.

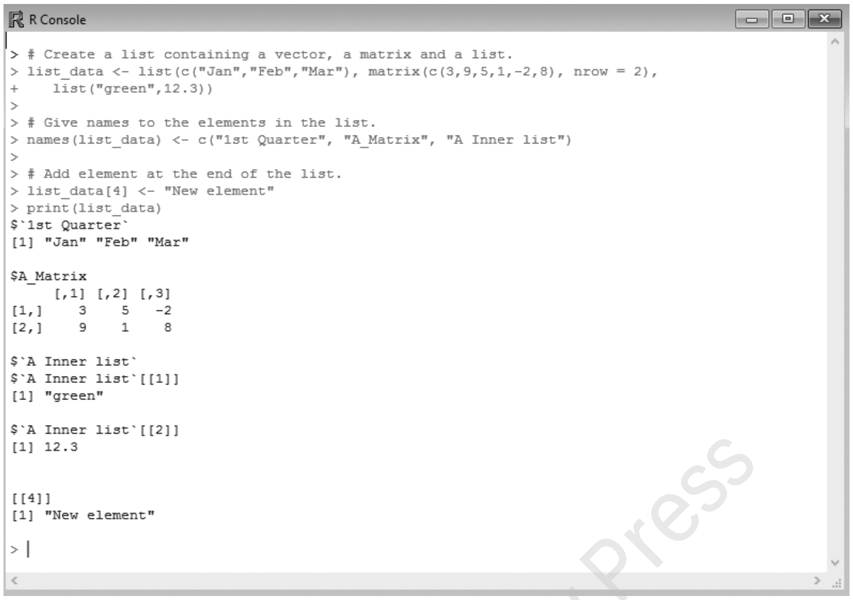
3.4 MANIPULATING LIST ELEMENTS

Addition, deletion, and updating operations can be performed on the list to manipulate the elements. Elements can be added and deleted only at the end of the list, whereas, the update operation can be performed on any element in the list irrespective of its position.

Example 3.2 Write the command in R console to add a new element at the end of the list and display the same.

Solution:

Figure 3.4 depicts the addition of an element to a list. Consider the example list created in Fig. 3.2, which consists of only three elements. To add the fourth element to this list, append the element (which is a string here) “New element” to the 4th indexed position of the list, that is, *list_data*[4]. To view the added element in the list, print the list and the four elements are displayed.



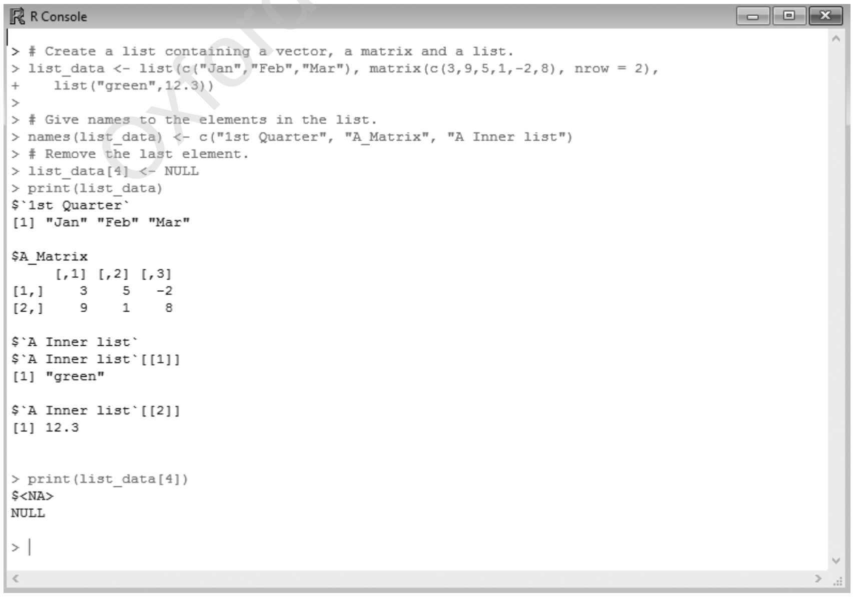
```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
>
> # Add element at the end of the list.
> list_data[4] <- "New element"
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$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


[[4]]
[1] "New element"

> |
```

Fig. 3.4 Adding an element to the list

Example 3.3 Write the command in R console to delete the fourth element from a list and display the resultant list.

Solution:

Here, in Fig. 3.5, the fourth element, “New element” can be deleted by setting its value to NULL. Printing *list_data* shows only three elements in the list and when the fourth element is tried to be printed, it shows NULL.

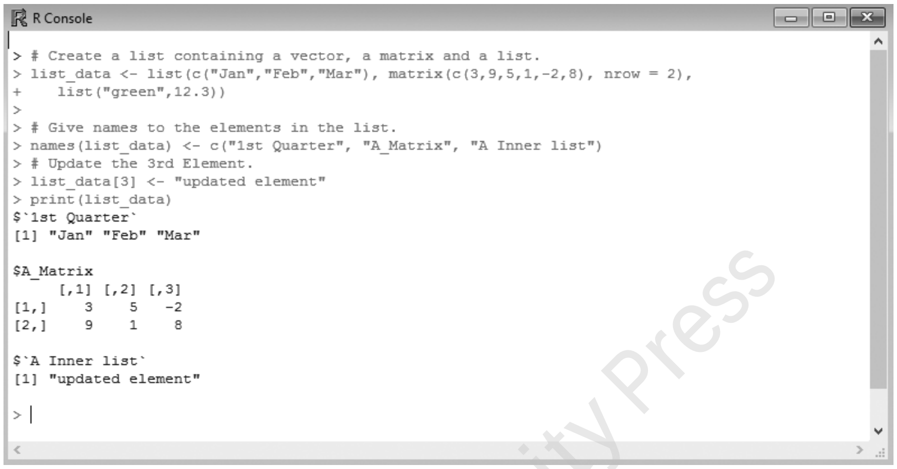


```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
> # Remove the last element.
> list_data[4] <- NULL
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$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


> print(list_data[4])
$<NA>
NULL

> |
```

Fig. 3.5 Deleting an element from a list

Example 3.4 Write the command in R console to update the third element of the list and display the resultant list.

Solution:
In Fig. 3.6, we update the third element of *list_data* as “updated element”. This is then checked by printing the list.



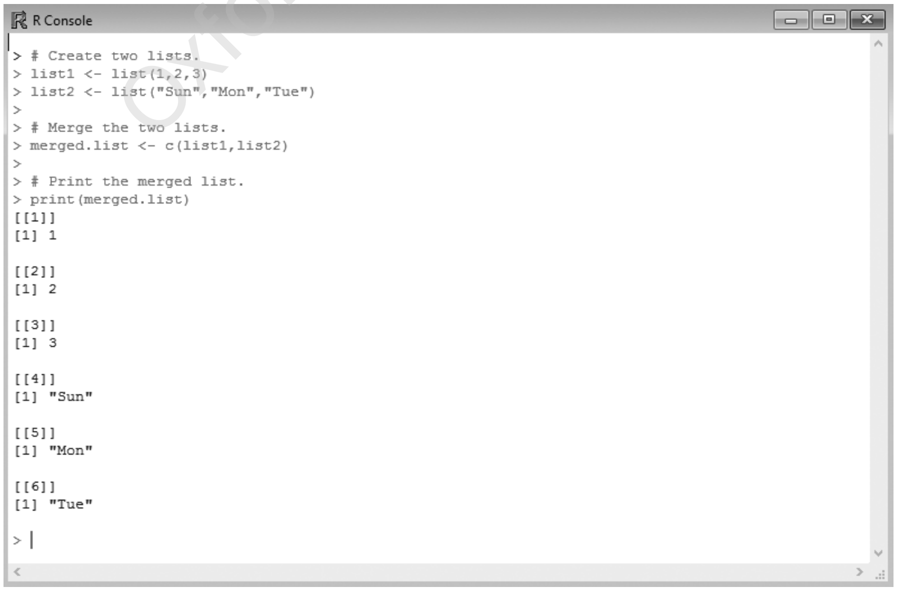
```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+    list("green",12.3))
>
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
> # Update the 3rd Element.
> list_data[3] <- "updated element"
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

$`A Inner list`
[1] "updated element"

> |
```

Fig. 3.6 Updating a list element

3.5 MERGING LISTS

Multiple lists can be merged by placing all those lists in a single *list*() function.

Figure 3.7 shows two lists *list1* and *list2* created using the *list*() function. Now placing them in a single *list*() function gives a merged list called *merged.list*, which contains the elements of both the lists placed sequentially.



```
> # Create two lists.
> list1 <- list(1,2,3)
> list2 <- list("Sun","Mon","Tue")
>
> # Merge the two lists.
> merged.list <- c(list1,list2)
>
> # Print the merged list.
> print(merged.list)
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"

[[5]]
[1] "Mon"

[[6]]
[1] "Tue"

> |
```

Fig. 3.7 Merging two lists

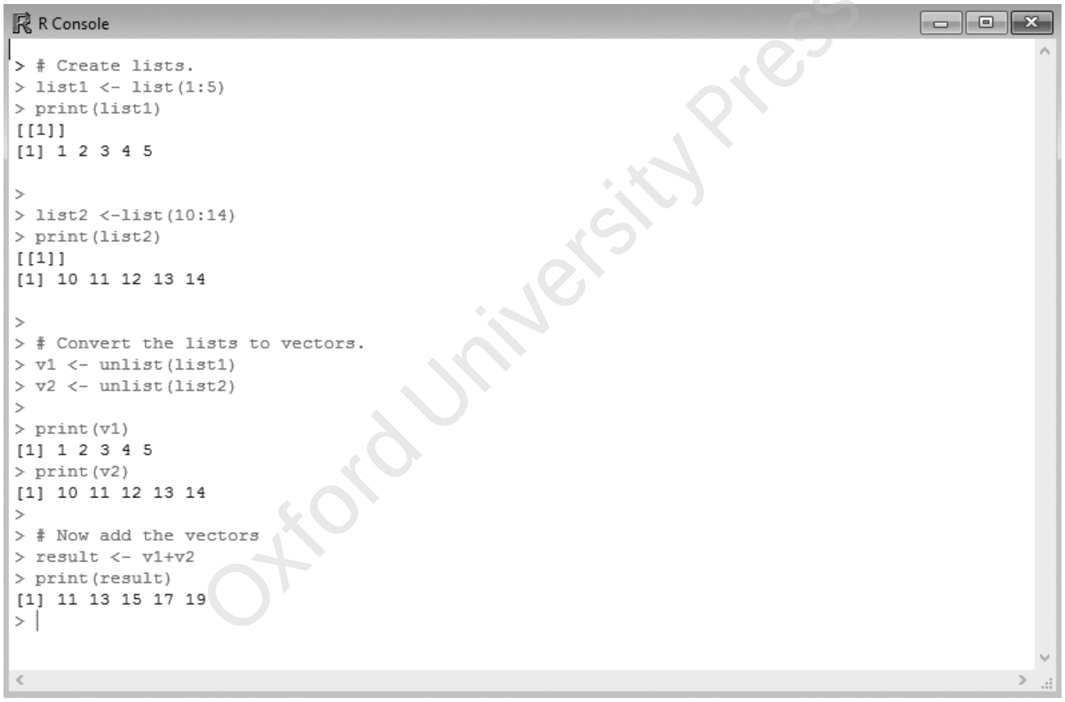
3.6 CONVERTING LISTS TO VECTORS

There are some special cases where lists are converted to vectors for further manipulation of the data elements. This is done because in addition to the basic operations of add, delete, and update, there are other arithmetic operations that can be easily applied when the operands are in vector form. For this conversion, a special function called *unlist*() is used; this function takes the input as list and produces vectors as depicted in detail in Example 3.5.

Example 3.5 Write the command in R console to create two lists, each containing 5 elements. Convert the lists into vectors and perform addition on the two vectors. Display the resultant vector.

Solution:

As shown in Fig. 3.8, two lists namely, *list1* and *list2* are initially created. Later with the help of the *unlist*() function, both the lists are converted to vectors named v1 and v2. Addition, subtraction, and multiplication of two vectors can be performed, which was not possible if the data was in the form of a list.



```
> # Create lists.
> list1 <- list(1:5)
> print(list1)
[[1]]
[1] 1 2 3 4 5

>
> list2 <-list(10:14)
> print(list2)
[[1]]
[1] 10 11 12 13 14

>
> # Convert the lists to vectors.
> v1 <- unlist(list1)
> v2 <- unlist(list2)
>
> print(v1)
[1] 1 2 3 4 5
> print(v2)
[1] 10 11 12 13 14
>
> # Now add the vectors
> result <- v1+v2
> print(result)
[1] 11 13 15 17 19
> |
```

Fig. 3.8 Converting lists to vectors

Programming Questions

3.1 Perform the following operations using lists:

(a) Construct a list, named *my_list* that contains the variables *my_vector*, *my_matrix* and *my_factor* as list components.

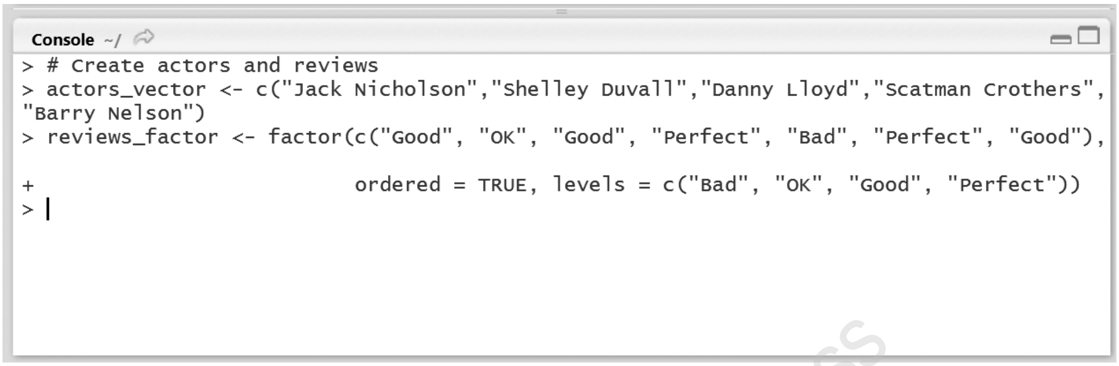
(b) Construct a list, named *my_super_list*, which now contains the four predefined variables listed.

(c) Print the structure of *my_super_list* with the *str*() function.

(d) Change the code that built *my_list* by adding names to the components. Use the names mat, vec, and fac, respectively, for *my_matrix*, *my_vector*, *my_factor.*

(e) Print the list to the console and inspect the output.

3.2 Being a huge movie fan, a user decides to start storing information on good movies with the help of lists. Create the variable *shining_list*. The list contains the movie title as *title*, then the actor names as *actors*, and finally the review scores factor as *reviews*. Pay attention to naming the objects correctly.



```
> # Create actors and reviews
> actors_vector <- c("Jack Nicholson","Shelley Duvall","Danny Lloyd","Scatman Crothers",
"Barry Nelson")
> reviews_factor <- factor(c("Good", "OK", "Good", "Perfect", "Bad", "Perfect", "Good"),

+                          ordered = TRUE, levels = c("Bad", "OK", "Good", "Perfect"))
> |
```

3.3 Create a list *lst* that contains the fifth element from the top and the entire fourth column of prop. Do not name this list.

3.4 Create a new list, *skills*, which contains top, cont, prop, and lst. Name the list elements *topics*, *context*, *properties*, and *list_info*, respectively. Users can either choose to first create the list with *list*() and name the list elements later using *names*() or can also build a named list in a single command. Use the *str*() function on *skills* to display the structure.

Concept Assessment Questions

3.1 What is the advantage of using a list?
  (a) Lists perform automatic coercion when confronted with elements of different types.
  (b) Unlike vectors and matrices, lists can contain all kinds of R objects.
  (c) Performing arithmetic operations with lists is easier compare to with matrices.
  (d) Lists perform automatic summaries for every row and column.

3.2 Which function should users use to display the structure of a list in R?

3.3 Which of the following data objects can be stored in a list?
  (a) Vectors (b) Matrices
  (c) Lists

3.4 Which of the following symbols allow users to extract a single element from an unnamed list.
  (a) [] (b) [[]]
  (c) $ (d) ( )

3.5 Which two of the following statements about selecting single elements from a list are true?
  (a) Users can select single elements from a named list by making use of subsetting by indices and double square brackets.
  (b) Users can select single elements from a named list by making use of subsetting by logicals.
  (c) Users can select single elements from a named list by making use of subsetting by name and single squared brackets.
  (d) Users can select single elements from a named list by making use of subsetting by the dollar sign.

3.6 Which two notations should users use to add a vector to a list?
  (a) [] (b) [[]]
  (c) $ (d) ( )
  (e) { }

3.7 What would the following code print?

```
> x <- list(1, “a”, TRUE, 1 + 4i)
  x
```

(a)
```
[[1]]
[1] 1
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1 + 4i
```
(b)
```
[[1]]
[1] 2
[[2]]
[1] "b"
[[3]]
[1] TRUE
[[4]]
[1] 1 + 4i
```
(c)
```
[[1]]
[1] 3
[[2]]
[1] "a"
[[3]]
[1] TRUE
[[4]]
[1] 1 + 4i
```
(d) All of these

3.8 Which of the following assignments is invalid?

(a) `> x <- fact(c("yes", "yes", "no', "yes", "no"))`

(b) `> x <- factor (c("yes", "yes", "no", "yes", "no"))`

(c) `> x <- factor (factor ("yes", "yes", "no", "yes", "no"))`

(d) None of these

3.9 What would the output of the following code be?

```
> x <- data.frame(foo = 1:4, bar =
C(T, T, F, F))
> ncol(x)
```

(a) 2 (b) 4

(c) 7 (d) All of these

3.10 Which of the following assignments is invalid?

(a) `> x <- list("Los Angeles" = 1, Boston = 2, London = 3)`

(b) `> names(x) <- c("New York", "Seattle", "Los Angeles")`

(c) `> name(x) <- c("New York", "Seattle", "Los Angeles")`

(d) None of these

3.11 What would the output of the following code be?

```
> x <- 1:3
> names (x)
```

(a) NULL (b) 1

(c) 2 (d) None of these

3.12 Suppose we have a list defined as x <- list(6, "c", "s", FALSE). What does x[[1]] give?

3.13 Write the output for each of the following statements:

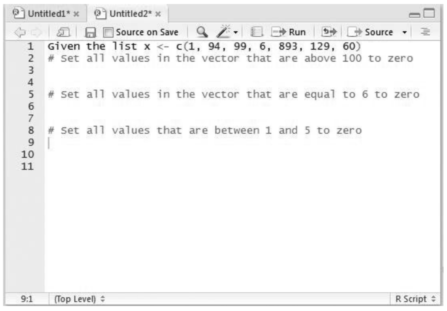
(a) `x <- c("a", "b", "c", "d", "e")`
   (i) print(x[1])
   (ii) print(x[2:4])
   (iii) print(x[x > "b"])
   (iv) u<- x > "a"

(b) `x<- list(foo = 5, bar = 2)`
   (i) print(x[[1])
   (ii) print(x$bar)
   (iii) print(x[["bar"]])

3.14 What is the difference among [, [[, and $ when applied to a list?

3.15 What is the difference between $ and [[?

3.16 Do as directed.



```
Given the list x <- c(1, 94, 99, 6, 893, 129, 60)
# Set all values in the vector that are above 100 to zero


# Set all values in the vector that are equal to 6 to zero


# Set all values that are between 1 and 5 to zero
```

3.17 Create a list that can be used in character subsetting with the following data:

`("m", "f", "u", "f", "f", "m", "m")`

`(m = "Male", f = "Female", u = NA)`

How would you convert the abbreviations to their original form?