

# **Learning Base R**

Second Edition

Lawrence M. Leemis  
Department of Mathematics  
William & Mary

**Library of Congress Cataloging-in-Publication Data**

Leemis, Lawrence M.

Learning Base R / Lawrence M. Leemis.

ISBN 978-0-9829174-5-9

Includes index.

1. R (Computer program language)

QA276.45.R3 L44 2022

© 2022 by Lawrence M. Leemis

All rights reserved. No part of this book may be reproduced in any form or by any means, without permission in writing from the publisher.

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the checking of the mathematics and computational techniques for correctness. The author and publisher make no warranty of any kind, expressed or implied, with regard to the mathematics and computational techniques contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the mathematics or computational techniques contained herein.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 978-0-9829174-5-9

For Arthur & Ivah



# Contents

- Preface . . . . . ix
- 1 Introducing R . . . . . 1**
- 2 R as a Calculator . . . . . 5**
  - 2.1 Order of operations . . . . . 5
  - 2.2 Number of digits displayed . . . . . 7
  - 2.3 Limits and undefined calculations . . . . . 8
  - 2.4 Integer divide and modulo . . . . . 9
- 3 Simple Objects . . . . . 12**
  - 3.1 Assigning numeric values to objects . . . . . 12
  - 3.2 Arithmetic operations on objects . . . . . 14
  - 3.3 Listing and removing objects . . . . . 14
  - 3.4 Selecting meaningful object names . . . . . 15
- 4 Vectors . . . . . 19**
  - 4.1 Creating a vector . . . . . 19
  - 4.2 Extracting elements of a vector . . . . . 23
  - 4.3 Vector arithmetic . . . . . 25
- 5 Matrices . . . . . 30**
  - 5.1 Creating a matrix . . . . . 30
  - 5.2 Extracting elements of a matrix . . . . . 33
  - 5.3 Matrix arithmetic . . . . . 34
- 6 Arrays . . . . . 38**
  - 6.1 Creating an array . . . . . 39
  - 6.2 Extracting elements of an array . . . . . 41
  - 6.3 Array arithmetic . . . . . 42
- 7 Built-In Functions . . . . . 44**
  - 7.1 Syntax, arguments, and elementary functions . . . . . 44
  - 7.2 Truncating and rounding . . . . . 49
  - 7.3 Sorting, ordering, and ranking . . . . . 50
  - 7.4 Properties of an object . . . . . 51
  - 7.5 Functionals . . . . . 53

<b>8</b>	<b>User-Written Functions</b>	<b>60</b>
8.1	Elementary functions . . . . .	61
8.2	Scoping . . . . .	65
8.3	Variable number of arguments . . . . .	67
<b>9</b>	<b>Utilities</b>	<b>72</b>
9.1	Managing the workspace . . . . .	72
9.2	Getting help . . . . .	74
9.3	Floating point representations . . . . .	78
<b>10</b>	<b>Complex Numbers</b>	<b>82</b>
10.1	Defining a complex number . . . . .	82
10.2	Operations on complex numbers . . . . .	83
10.3	Vectors, matrices, and arrays of complex numbers . . . . .	84
<b>11</b>	<b>Character Strings</b>	<b>88</b>
11.1	Simple character strings . . . . .	88
11.2	Automatic coercion . . . . .	90
11.3	Built-in objects of character strings . . . . .	90
11.4	Character string manipulation . . . . .	91
11.5	Names . . . . .	93
11.6	Factors . . . . .	94
<b>12</b>	<b>Logical Elements</b>	<b>101</b>
12.1	Operations on logical elements . . . . .	101
12.2	Using logical elements as subscripts . . . . .	104
<b>13</b>	<b>Relational Operators</b>	<b>107</b>
13.1	Relational operators applied to objects . . . . .	107
13.2	Conditional execution via <code>ifelse</code> . . . . .	111
13.3	Conditional execution via <code>switch</code> . . . . .	113
<b>14</b>	<b>Coercion</b>	<b>118</b>
14.1	The <code>is</code> family of functions . . . . .	118
14.2	The <code>as</code> family of functions . . . . .	120
<b>15</b>	<b>Lists</b>	<b>126</b>
15.1	Creating a list . . . . .	126
15.2	Extracting elements of a list . . . . .	127
15.3	Functions that operate on lists . . . . .	128
<b>16</b>	<b>Data Frames</b>	<b>133</b>
16.1	Creating a data frame . . . . .	133
16.2	Functions that operate on data frames . . . . .	134
16.3	Extracting elements of a data frame . . . . .	135

---

<b>17 Built-In Data Sets</b>	<b>141</b>
17.1 The iris data set . . . . .	141
17.2 Extracting elements of a data set . . . . .	144
17.3 A taxonomy of some built-in data sets . . . . .	147
<b>18 Input / Output</b>	<b>152</b>
18.1 Input . . . . .	152
18.2 Output . . . . .	157
<b>19 Probability</b>	<b>162</b>
19.1 Random numbers . . . . .	162
19.2 Binomial distribution . . . . .	163
19.3 Poisson distribution . . . . .	164
19.4 Uniform distribution . . . . .	166
19.5 Normal distribution . . . . .	167
19.6 Other distributions . . . . .	169
19.7 Random sampling . . . . .	170
<b>20 High-Level Graphics</b>	<b>175</b>
20.1 Univariate data . . . . .	175
20.2 Multivariate data . . . . .	180
20.3 Categorical data . . . . .	183
20.4 Time-series data . . . . .	184
<b>21 Custom Graphics</b>	<b>189</b>
21.1 Graphical issues . . . . .	189
21.2 Examples . . . . .	190
<b>22 Conditional Execution</b>	<b>209</b>
22.1 The if statement . . . . .	209
22.2 Applications . . . . .	212
<b>23 Iteration</b>	<b>218</b>
23.1 The while loop . . . . .	218
23.2 The for loop . . . . .	220
23.3 The repeat loop . . . . .	222
23.4 Algorithm development . . . . .	224
23.5 Debugging . . . . .	231
23.6 Optimization . . . . .	233
<b>24 Recursion</b>	<b>241</b>
24.1 Calculating factorials . . . . .	241
24.2 Binary search algorithm . . . . .	244
24.3 Tower of Hanoi . . . . .	246
<b>25 Simulation</b>	<b>250</b>
25.1 Random number generation . . . . .	250
25.2 Generating Bernoulli trials . . . . .	253
25.3 Monte Carlo simulation . . . . .	254

<b>26 Statistics</b>	<b>267</b>
26.1 Univariate data analysis . . . . .	267
26.2 Analysis of variance . . . . .	273
26.3 Regression . . . . .	279
26.4 Time series analysis . . . . .	293
<b>27 Linear Algebra</b>	<b>304</b>
27.1 Elementary operations . . . . .	306
27.2 Inverses and determinants . . . . .	313
27.3 Eigenvalues and eigenvectors . . . . .	316
27.4 Decompositions . . . . .	318
<b>28 Packages</b>	<b>325</b>
28.1 The big picture . . . . .	325
28.2 Packages attached to my current R session . . . . .	328
28.3 Packages installed on my computer . . . . .	329
28.4 Contributed packages . . . . .	335
<b>Index</b>	<b>344</b>

# Preface

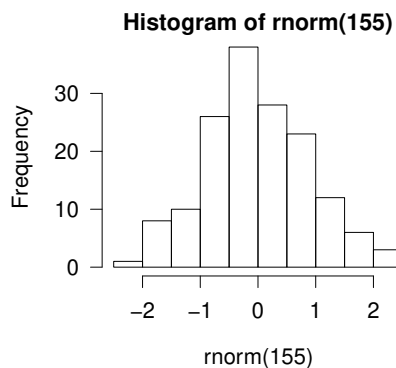
R is a free software package that includes a rich set of data structures, computational functions, graphical functions, statistical procedures, programming capability, and much more. This book is designed for novices to R, both those with and without prior programming experience. The emphasis here is on the base capability of R; extended capabilities via the use of packages are introduced in the final chapter.

The chapters in this text introduce R in small, bite-sized chunks, which take about 15 minutes apiece, on average, to digest. The best way to go through this text is to have R installed and running on a nearby laptop or desktop machine, try each command presented, then experiment a bit with other similar commands. Think of each chapter as a separate R session. All of the commands in this text are available on my home page at [www.math.wm.edu/~leemis](http://www.math.wm.edu/~leemis) in case you would like to save some typing. I have worked with R and its predecessors S and S-Plus for many years, and I have yet to crash the software, so feel free to experiment with reckless abandon.

Two small examples are provided here to illustrate how short R commands involving just a few keystrokes can result in output that would require significant programming in other languages. Consider the single R command

```
> hist(rnorm(155))
```

The greater-than symbol (>) at the beginning of the line is a prompt that awaits an R command from the user. After the R command `hist(rnorm(155))` is keyed in and the return key is pressed, R opens a graphics window displaying the histogram shown below.



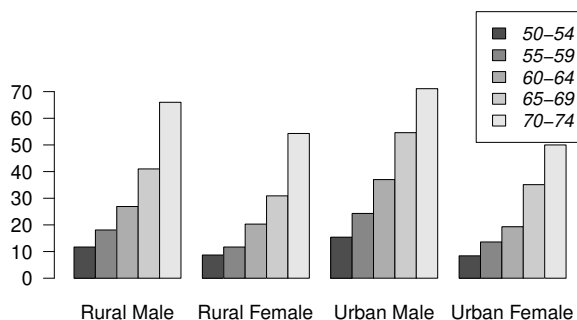
The function name `hist` is short for *histogram*. The argument associated with `hist` is the data to be plotted in the histogram. Rather than using an actual data set collected in an experiment, the data is given by `rnorm(155)`, which is 155 random variates (the number of random variates was selected arbitrarily) drawn from a standard normal population.

Not surprisingly, the histogram shows a roughly bell-shaped distribution of the data values; its shape is bumpy due to random sampling variability. There are dozens of default decisions that were made in constructing this histogram. For example, there are 10 cells to hold the data values, the tick marks extend out of the plot, the histogram bars are not shaded, the heights of the bars are the counts of the number of data values falling in the cells, and a main title and axis labels are included. All of these decisions are easily modified to produce a custom plot.

Rather than using simulated data values, a second example uses a real data set. The R command

```
> barplot(VADeaths, beside = TRUE, legend = TRUE)
```

produces a bar chart, complete with a legend, of the `VADeaths` data set built into R. The bar chart appears in a graphics window and is shown below. Once again, R makes several decisions internally to produce the graphic. R selects a scale for the vertical axis, and plots bars of appropriate heights. R includes tick marks and associated labels. R even found some room in the upper-right-hand corner of the graphic and placed the requested legend there. R chose to shade the bars in the bar chart. All of these are easily-modified default decisions.



It is not clear what is in the data set `VADeaths`. Typing

```
> VADeaths
```

shows the data values and typing

```
> help(VADeaths)
```

gives information concerning the data set, revealing that the heights of the bars are annual death rates per 1000 residents, stratified by age, gender, and domicile, in Virginia in the 1940s.

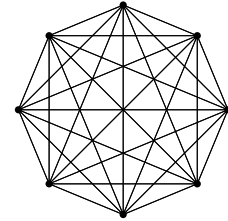
My thanks goes to Russell Atkinson, Barry Lawson, and Tanujit Dey for ideas and brainstorming, and to Nadia Aly, Taylor Arnold, Daniel Block, Rob Downs, Anthony Finch, Jamie Joseph, Kristina Kelly, Jace King, Samantha King, Larson Lasek, Arthur Leemis, Bob Lewis, Seth Lyles, Robert Marmorstein, Barry Nelson, Bob Noonan, Scott Percic, Mike Pozulp, Melissa Tilashalski, and Andrew Turscak for looking the text over prior to publication. Thanks also to Daniel McGibney and Raghu Pasupathy for class testing the text. Special thanks goes to Diane Evans, Jamie Joseph, Bob Lewis, and Yuxin Qin, who read an entire draft of the second edition, providing thousands of edits for the final version. I also thank Lindsey Leemis for the handsome cover design. Finally, thanks to the designers and contributors to S, S-Plus, and R, and to Don Knuth for  $\text{\TeX}$ . Any suggestions for the improvement of the text will be gratefully received. Enjoy R!

Williamsburg, VA

Larry Leemis  
November, 2021

# Chapter 1

## Introducing R



R is a programming language that has steadily grown in popularity over the past few decades. Regardless of whether you work in industry, academics, or government, or if you just use the language for personal use, you will always have access to the R language. So why use R? There are at least eight answers to this question.

1. R is open source software. It can be downloaded at no cost onto a desktop or a laptop computer.
2. R is capable of performing numerical calculations on scalars, vectors, and matrices. It can be used as a high-powered calculator.
3. R has built-in functions for performing probability and statistical calculations. R can perform simulation experiments using these functions.
4. R has extensive graphical capabilities. R can generate production-level graphics using built-in graphical functions. These flexible functions can generate customized graphics.
5. R is a programming language. R allows the user to execute several types of loops for iteration and supports conditional execution of code.
6. R is reproducible. R commands posted on the internet, for example, can simply be copied, pasted, and executed in your current R session.
7. R has a vast array of support resources, including textbooks, on-line tutorials, on-line answers to specific questions, an academic journal, and the annual international *UseR!* conference.
8. R has thousands of contributors who have written R code contained in *packages*, which continue to extend the language in the same way that apps have transformed handheld devices.

The title of this book, *Learning Base R*, emphasizes that the portion of R introduced here is the “base” language, which is what you encounter when you initially enter an R session. The final chapter considers *packages*, which substantially extend the capability of the base language.

The R language, originally known as S, was developed in 1976 at Bell Labs by John Chambers and his colleagues. S-Plus is the current commercial version of S. In 1993, the free, open source R language was first developed by Ross Ihaka and Robert Gentleman (notice the common first initial) from the University of Auckland. R and S have a very large overlap of capabilities. If you know one, you essentially know the other. The differences are minor. One advantage to R over S, however,

is the vast number of packages that have been written in R that are capable of extending the base language. These packages can be quite useful in certain applications.

Many R users prefer to use R in an *integrated development environment*, often abbreviated IDE. One popular IDE for R is RStudio, which makes the process of writing an R program more convenient than working in the native R language. Several panes open when you enter RStudio. The lower-left pane usually corresponds to a native R session. This book applies equally well to native R and RStudio. If you are new to R and unsure which platform to use, try RStudio for its convenience in developing R programs.

The orientation for R is as follows. R is a vector-based language that uses vectors as its primary data structure. R has a command-line orientation rather than a Graphical User Interface (GUI) menu orientation. Although this might seem a bit antiquated at first, the size and capability of the language force this orientation. The focus here will be on the syntax: the rules for issuing R commands. There are plenty of other R books on the market that are much more encyclopedic in nature or cover specific applications. This book is designed to be a quick introduction to the language for R novices. More advanced books on R will be much easier to read once you master the basics presented here.

You will notice that R tends to favor short, abbreviated variable and function names, which is helpful on your fingers in terms of saving keystrokes. This is a heritage from the C language and the Unix environment that was present at Bell Labs in the 1970s.

The R language can be located by doing a web search on the letter R in a browser. Alternatively, you can go directly to the website <http://www.r-project.org>. Once you get to the website you should pick a nearby mirror site and install the binary version. You will need to choose an appropriate platform: Windows, Mac, or Linux. You might also need to choose between the 32-bit and 64-bit version of R. For your first installation, it is probably best to select the default location for installing R. Installing R is similar to installing any other type of software. A license agreement must be accepted and you will want to install the most recent base version of the software. In addition to the version number, there is an associated release name, such as “Lost Library Book,” “You Stupid Darkness,” “Bug in Your Hair,” or “Great Pumpkin.” The source of these curious release names is unclear, but they seem to reflect the season of the release, and many are taken from Charles Schulz’s *Peanuts* comic strips and films. R is a fairly large language, so installing it typically takes a few minutes. On most platforms, an icon with the letter R will appear on your desktop after downloading and installation. To launch R or RStudio, double-click the appropriate icon on your desktop. This will bring up a window which contains a greater-than prompt, which looks like this:

```
>
```

The prompt indicates that R is waiting for an R command from you. The remainder of this book describes what you can type after the prompt and the associated results. If you type any valid R command at this point and press the return key, R will display the output. If a command is too long for a single line, it is fine to press the return key and continue the command over several lines. A + prompt automatically appears to remind you that you are completing a command from the previous line. If a command requires significant processing time, there will be a time delay for processing before a new prompt appears. This orientation of having a prompt which awaits an R command from you is known as the *command line interface*. The output from an R command might vary slightly depending on the platform (Windows, Mac, or Linux) on which R is running.

When your R session is finished, you can quit R by typing `q()`, for quit, at the prompt. A dialog box will open that asks: Save workspace image? [y/n/c]. This determines whether R saves the objects that you have created during the R session for a future R session. After responding to that question with y or n, for yes or no, your R session is completed. Pressing c for *cancel* allows the R session to continue.

Experienced programmers who are used to working with compiled languages may find R's orientation foreign. The practice of submitting a command, viewing the results, then perhaps submitting a subsequent command based on the results is not the way compiled languages are generally used. This orientation is largely a result of the roots of R in exploratory data analysis (sometimes abbreviated EDA). One preliminary look at the data often guides subsequent analysis techniques. It is easy to execute a series of R commands that are stored in an external file if the exploratory data analysis approach is not appropriate in a particular setting. Also, R stores its matrices in a column-major orientation, as opposed to the more standard row-major orientation. This is also an artifact of R's roots in exploratory data analysis and the associated data structures required to efficiently store data sets.

R is an interpreted rather than a compiled language, which means that each R command must be interpreted and then executed. Therefore, run times for R code will be a bit slower than comparable code in a compiled language. Some speedup is possible, however, with vector-based programming, which will be described later in the text.

*Comments* are useful for documenting the purpose of a particular R command or a group of R commands either for yourself or for someone else who might use your code. A comment consists of text that is ignored by R as it processes your R input command. When R encounters the # symbol, it ignores all subsequent input on that line.

As you begin to wade into R, there are several tricks that can make your R session more efficient. Here are five such tricks.

1. You can browse through previous R commands using the up arrow. If you encounter a command that you would like to alter, you can use the destructive backspace, or the nondestructive backspace (the left arrow) to edit the R command. Pressing the enter key executes the modified R command even if the cursor is not at the end of the line.
2. Tab completion is another way to save keystrokes. For example, if you type

```
> VA
```

at the R prompt and then press the tab key, R recognizes that `VADeaths` is the only object that begins with the letters `VA`, so it completes the name of the object.

3. Sometimes your R session can get cluttered. Typing control- $\ell$  clears the screen in both native R and in the console pane of RStudio.
4. Several R commands can be executed at once. In RStudio, the commands are keyed into the code development pane. Pressing the *Run* button executes the current line or the highlighted lines. In native R, enter R by double-clicking on the R icon. Using the *File* drop-down menu, click on *New Document*, which opens a window for the R commands. After the R commands are keyed in, command-enter executes the current line or the highlighted lines.
5. R code that you encounter on the internet can be copied and pasted into your R session to be executed. As with anything taken from the internet, proceed with caution.

The R language does not have a natural way to sequence the introduction of the various topics. I have decided to sequence the subsequent chapters in the following fashion.

- Chapters 2 and 3 introduce elementary arithmetic operations and named storage in R.
- Chapters 4, 5, and 6 introduce three elementary data structures: vectors, matrices, and arrays. In these early chapters, these data structures are filled with numeric elements.

- Chapters 7 and 8 introduce built-in and user-written functions.
- Chapter 9 introduces some useful utilities.
- Chapters 10, 11, and 12 introduce three other types of elements that can be stored in a data structure: complex numbers, character strings, and logical elements.
- Chapters 13 and 14 introduce methods for comparing elements with relational operators and coercing elements to a particular data type.
- Chapters 15 and 16 introduce two advanced data structures: lists and data frames.
- Chapter 17 surveys some data sets that are built into R.
- Chapter 18 introduces input and output functions that interface R with the outside world.
- Chapter 19 introduces functions that are useful in probability calculations.
- Chapters 20 and 21 introduce R's graphical capabilities.
- Chapters 22, 23, and 24 introduce R's programming capabilities.
- Chapter 25 shows how to conduct simulations in R.
- Chapter 26 surveys R's capability for performing statistical inference.
- Chapter 27 introduces linear algebra functions that are built into R.
- Chapter 28 shows how to extend R's base capability by using packages.

You will find R to be a very efficient language for performing both simple and complex calculations. R can be used for analyzing data, performing simulations, generating graphics, etc. The value of a language like R was recognized long ago.

Civilization advances by extending the number of important operations which we can perform without thinking about them. —Alfred North Whitehead (1861–1947)

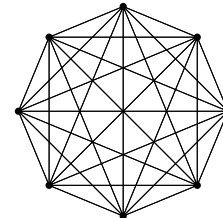
The next chapter introduces R in its simplest form—as a calculator.

## Exercises

- 1.1** Download R from the website <http://www.r-project.org> onto your desktop or laptop computer. Optionally, download RStudio from the website <https://www.rstudio.com>. If you decide to use RStudio instead of native R, you must also download R.
- 1.2** The Comprehensive R Archive Network (CRAN) website <http://cran.r-project.org> contains links to various sites concerning the R language. Browse some of the online documentation for R and write a paragraph describing a site that you found interesting.
- 1.3** Launch an R session and practice with tab completion.
- 1.4** Launch an R session and practice using the up arrow to repeat previous commands.

## Chapter 2

# R as a Calculator



R can be used as a calculator. This chapter (a) outlines rules associated with the order in which R executes basic arithmetic operations, (b) introduces a function that controls the number of digits that R uses to display the results of a calculation, (c) shows how R handles an extreme calculation like  $1/0$ , and (d) introduces the integer-divide and modulo operators.

In the examples given throughout this text, R commands that you type will be set in monospace font following the `>` prompt. The response from R will be given on the next line or lines, also set in monospace font. This is exactly what you will see in an R session.

### 2.1 Order of operations

We begin with simple arithmetic operations. The order in which arithmetic operations are performed is consistent with the PEMDAS convention. This convention implies that *parentheses* get highest priority, followed by *exponentiation*, followed by *multiplication* and *division*, followed by *addition* and *subtraction*. The first example of using R as a calculator is given below.

```
> 2 + 9 * 4                # PEMDAS convention; spaces around operators
[1] 38
```

No equal sign is necessary; pressing the return key serves that role. The response from R is `[1] 38`, which means that R performed the multiplication before the addition per the PEMDAS convention, giving a result of 38. The `[1]` that appears before the 38 indicates that the result is a vector of length one. More details concerning vectors will be given in Chapter 4. The `[1]` can be ignored for now. The text after the `#` is a comment and has been added to highlight the concept being presented. It is good programming practice to surround the operators with a single space for readability. This is not necessary, however, and it is perfectly acceptable to key in the R command as

```
> 2+9*4                    # no spaces around operators; it still works
[1] 38
```

But you could imagine how difficult this R command would be to read if it was 30 or 40 characters long. Surrounding operators with spaces and placing a space after a comma as you key in R commands is a good habit to develop.

The exponentiation operator is the caret symbol, `^`, as illustrated by

```
> 4 + 3 / 10 ^ 2      # exponentiation first, then division, then addition
[1] 4.03
```

The slashes on the zeros are to distinguish between a zero and an uppercase O. Parentheses can be used to alter the order of operations.

```
> 4 + (3 / 10) ^ 2    # division first, then exponentiation, then addition
[1] 4.09
> (4 + 3) / 10 ^ 2    # addition first, then exponentiation, then division
[1] 0.07
> (4 + 3 / 10) ^ 2    # division first, then addition, then exponentiation
[1] 18.49
> ((4 + 3) / 10) ^ 2  # addition first, then division, then exponentiation
[1] 0.49
```

If you press the return key prior to completing a command, the usual greater than prompt will be replaced by a + prompt, which is R asking you to continue the command.

```
> 2 - 3 *             # + prompt for more input
+ 7                   # completed command 2 - 3 * 7
[1] -19
```

An R command that traverses several lines in this fashion is appropriate for a particularly-long calculation. Now consider computing  $1/4$ .

```
> 1 / 4               # the display suppresses trailing zeros
[1] 0.25
```

R calculates one-fourth and displays it as the decimal equivalent 0.25, suppressing the trailing zeros. Now consider computing  $1/3$ .

```
> 1 / 3               # default seven rounded digits displayed
[1] 0.3333333
```

R must make a decision on how many digits to display. The default in R is to display seven digits. This can be altered, however, as will be shown in the next section.

Notice that R does not include commas to separate groups of three digits when displaying the result of a calculation.

```
> 8 * 10 ^ 4          # exponentiation first
[1] 80000
```

The same is true for large numbers in an R command. Consider the calculation of a large number.

```
> 1111111 * 1111111  # the result is displayed in scientific notation
[1] 1.234568e+12
```

The result of this calculation is expressed in scientific notation:  $1.234568 \cdot 10^{12}$ . With only seven digits displayed, it is not possible to know the exact value of the product. Large and small numbers can be keyed into R in the same fashion in which they are displayed. Avogadro's number from chemistry,  $6.02214 \cdot 10^{23}$ , named after Italian chemist Amedeo Avogadro (1776–1856), for example, is keyed into R as

```
> 6.02214e+23        # Avogadro's number in scientific notation
[1] 6.02214e+23
```

## 2.2 Number of digits displayed

By default, R displays seven rounded digits, although more digits are stored internally. As an illustration, the built-in constant  $\pi$  is stored in the object named `pi`.

```
> pi # display pi to seven rounded digits
[1] 3.141593
```

The `options` function has several optional arguments, which are placed in parentheses and separated by commas if there are multiple arguments. One of these arguments controls the number of digits that are displayed when R displays the result of a calculation. The call to the `options` function below with the `digits` argument changes the default of displaying seven digits to displaying ten digits in all subsequent calculations.

```
> options(digits = 10) # display ten digits
> pi # display pi to ten rounded digits
[1] 3.141592654
```

R stores constants as floating-point numbers to machine precision internally. The largest number of digits that can be displayed is 22. So asking for 50 digits in `options` results in an “error message.” R sets the error message in bold monospace font, presumably to grab the user’s attention.

```
> options(digits = 50) # display 50 digits
Error in options(digits = 50) :
invalid 'digits' parameter, allowed 0...22
> options(digits = 16) # display 16 digits
> pi # display pi to 16 rounded digits
[1] 3.141592653589793
```

Now that more digits are displayed, the multiplication from the previous section can be calculated exactly and will not be displayed in scientific notation.

```
> 1111111 * 1111111 # the product is now exact
[1] 1234567654321
```

Of course a large number raised to a large power will need to be expressed in scientific notation, but this time with more digits displayed.

```
> 789 ^ 10 # expressed in scientific notation
[1] 9.349104886236698e+28
```

The exact value displayed for this calculation might be machine dependent.

R users should be aware that some numbers are not stored exactly internally because they are stored internally in binary. For example, the constant 17.046875 is stored exactly by R because it has an exact binary representation. However, the constant  $6.02214 \cdot 10^{23}$  is *not* stored exactly in R because it does not have an exact binary representation.

```
> options(digits = 22)
> 17.046875 # stored exactly
[1] 17.046875
> 6.02214e23 # not stored exactly
[1] 6022139999999999969067008
```

Further details concerning the internal representation of numeric values in R are given in Chapter 9. The reciprocal of seventeen raised to the seventh power is a small number that will also be stored in scientific notation.

```
> 1 / 17 ^ 7          # a small number
[1] 2.437011341604645826792e-09
```

## 2.3 Limits and undefined calculations

R has special procedures for handling limits and undefined calculations such as division by zero or taking the square root of a negative number. First, try dividing one by zero.

```
> 1 / 0              # ratio of one to zero
[1] Inf
```

The reserved R object `Inf` represents infinity. Calculations involving `Inf` can also be executed.

```
> 1 / Inf            # ratio of one to infinity
[1] 0
```

The ratio of one to infinity is zero, which is consistent with intuition. Calculations involving `Inf` can be thought of as the associated mathematical limit. Some other calculations with `Inf` are

```
> Inf + Inf         # infinity plus infinity
[1] Inf
> Inf - Inf         # infinity minus infinity
[1] NaN
> Inf / Inf         # ratio of infinity to infinity
[1] NaN
```

The output from R on the last two calculations is another reserved object `NaN`, which stands for “Not a Number.” When R calculates the ratio of negative one to zero, it displays negative infinity.

```
> -1 / 0            # ratio of negative one to zero
[1] -Inf
```

When R encounters the ratio of zero to zero, which is undefined, it returns `NaN`.

```
> 0 / 0             # ratio of zero to zero
[1] NaN
```

When zero is raised to the zero power, the result is one.

```
> 0 ^ 0            # zero raised to the zero power
[1] 1
```

As expected, the square root of a negative number is undefined.

```
> (-9) ^ (1 / 2)   # square root of negative nine
[1] NaN
```

Chapter 10 introduces complex numbers in R. Taking the square root of a negative number is permitted when using complex numbers.

## 2.4 Integer divide and modulo

There are two other operators that are of interest in calculator mode. The first gives the integer part of a fraction.

```
> 37 %% 5          # integer part of a fraction
[1] 7
```

This operator is often known as “integer divide.” The second operator, known as the “modulo” operator, gives the remainder associated with the division of two integers.

```
> 37 % 5          # modulo (remainder) function
[1] 2
```

This concludes this brief introduction to R as a calculator. The introduction to R as a calculator is incomplete, however, because calculators have buttons for calculating square roots, logarithms, trigonometric functions, etc. R can perform calculations of this type as well, but the discussion of these topics is postponed until Chapter 7. The table given next summarizes the arithmetic operators, in decreasing priority order, described in this chapter.

operator	description
$\wedge$	exponentiation
$\%/\%, \%\%$	integer divide, modulo
$*, /$	multiplication, division
$+, -$	addition, subtraction

Parentheses can be used to alter the order of operations from the default. R uses `Inf` to represent infinity, which is generated by, for example, `1/0`. R uses `NaN` to represent “not a number,” which is generated by, for example, `0/0`.

Some of the exercises in this book ask for the output generated by a single R command or several R commands. You are encouraged to guess what will appear before entering an R session to find out what will be displayed. Anticipating the correct output is a good sign that you are learning the R language.

The next chapter defines the notion of an “object” in R. Using objects is the key to extending the capability of R well beyond that of a calculator.

## Exercises

**2.1** Write an R command to compute

$$100 \left( 1 + \frac{0.05}{12} \right)^{24},$$

which is the value of \$100 invested at an annual percentage rate (often abbreviated APR) of 5% after two years with monthly compounding.

**2.2** What is displayed by the following R command? (Guess before computing this in R.)

```
> 5 %% 2
```

- 2.3** What is the remainder when 3333 is divided by 222?
- 2.4** Use R to investigate the behavior of  $(1 + 1/n)^n$  for large, positive integer values of  $n$ .
- 2.5** Write R commands to calculate the golden ratio

$$\phi = \frac{1 + \sqrt{5}}{2}$$

to ten-digit accuracy.

- 2.6** The speed of light is 299,792,458 meters per second. Use R to calculate the amount of energy (the units will be Joules) in 10 kilograms of mass using Albert Einstein's

$$E = mc^2$$

formula, where  $m$  is the mass,  $c$  is the speed of light, and  $E$  is the energy.

- 2.7** Newton's law of universal gravitation states that two bodies attract each other with force

$$F = \frac{Gm_1m_2}{r^2},$$

where  $m_1$  and  $m_2$  are the masses of the two objects,  $r$  is the distance between the centers of mass of the two objects, and  $G = 6.67 \cdot 10^{-11}$  is the gravitational constant. Use R to calculate the force between the earth, with mass  $m_1 = 5.97 \cdot 10^{24}$  kilograms, and the moon, with mass  $m_2 = 7.35 \cdot 10^{22}$  kilograms. The average distance between the earth and the moon is  $r = 384,000,000$  meters.

- 2.8** The sine of  $\pi/96$  is

$$\sin\left(\frac{\pi}{96}\right) = \frac{1}{2} \sqrt{2 - \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{3}}}}}$$

Use R to calculate  $\sin(\pi/96)$ .

- 2.9** The sine of  $\pi/96$  is

$$\sin\left(\frac{\pi}{96}\right) = \frac{1}{2} \sqrt{2 - \sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{3}}}}}$$

How many terms in the Taylor series approximation

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

are necessary to achieve ten-digit accuracy to approximate  $\sin(\pi/96)$ ?

- 2.10** The simple arithmetic problem  $8 \div 2(2 + 2)$  lit up the internet over the summer of 2019. Compute this value using R. Do an internet search on this problem that went viral and decide whether you believe the solution that you get from R.

**2.11** The first ten numbers of the *Fibonacci sequence* are

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55.$$

If the first two numbers in this sequence are defined to be

$$x_1 = 1 \quad \text{and} \quad x_2 = 1,$$

then subsequent numbers in the sequence are found by the recursive formula

$$x_i = x_{i-1} + x_{i-2},$$

for  $i = 3, 4, \dots$ . Of course calculating  $x_{50}$ , for example, with pencil and paper, requires some significant arithmetic. Binet's formula allows us to calculate the value of any term in the sequence without calculating all of the intermediate sums:

$$x_n = \frac{(1 + \sqrt{5})^n - (1 - \sqrt{5})^n}{2^n \sqrt{5}}$$

for  $n = 1, 2, \dots$ . Write an R command to calculate  $x_{50}$ .

**2.12** Consider the quadratic equation

$$ax^2 + bx + c = 0,$$

where  $a$ ,  $b$ , and  $c$  are real-valued constants and  $a \neq 0$ . A quadratic equation can have zero, one, or two real-valued solutions. A quadratic equation can be solved by factoring, completing the square, or the quadratic formula. The quadratic formula gives the solutions

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

The solution(s) are real if the discriminant  $b^2 - 4ac$  is greater than or equal to zero. Calculate the real-valued solutions of the quadratic equation

$$x^2 + 3x + 2 = 0$$

via the quadratic formula using two R commands.