An Introduction to Computational Probability and Statistics with R

Bee Leng Lee

Contents

Part I

An Introduction to R

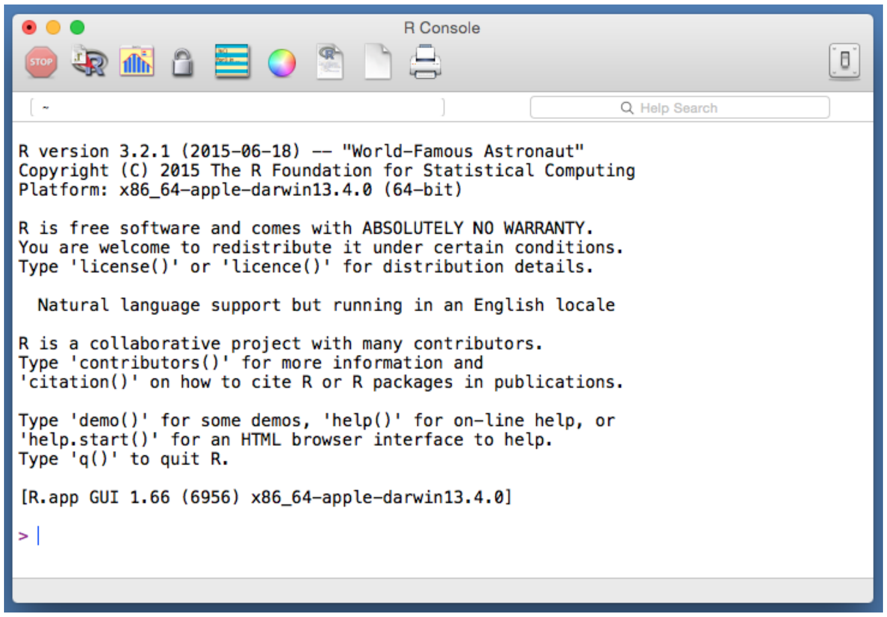
Chapter 1

Getting Started

R is a programming language and comprehensive statistical platform for data exploration and analysis. It is free and open source, which means anyone can download and use the latest version of the software free of charge, and the source code can be studied and modified without any restriction. Yet the functionality of R rivals commercial packages. Organizations and companies such as the FDA, Facebook and Google use R on a daily basis [10]. The minimalist interface of R, however, can be daunting to beginning users, especially those who are accustomed to the point-and-click interfaces offered by commercial packages. The purpose of this chapter is to orient beginning users. An up-to-date version of R for various computing platforms can be downloaded from the Comprehensive R Archive Network (CRAN) at `http://cran.r-project.org/`. We hasten to point out that while the R graphical user interface (RGui) appears a little different on each platform, there is no substantive difference otherwise.

1.1 The R Console

When R starts, a window similar to that shown in Figure 1.1 is presented. By default the window, called the R *console*, displays some information about R and a *command prompt* that is represented as a greater-than symbol (">"). The command prompt invites the user to type commands into R. When the user completes a command and presses return or enter, the R *interpreter* evaluates the command and displays the result in the console or a new window



```
R version 3.2.1 (2015-06-18) -- "World-Famous Astronaut"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.66 (6956) x86_64-apple-darwin13.4.0]

> 
```

Figure 1.1: The R console on Mac OS X.

whenever appropriate.

To illustrate the command-line interface, we use R as a calculator to evaluate the sum of 1, 2 and 3:

```
> 1 + 2 + 3
[1] 6
```

The first line, excluding the command prompt, shows the command typed by the user. The `[1]` preceding the result in the second line indicates that `6` is the first element of the result—even when the result comprises one element. If a command is incomplete when return or enter is pressed, R will display the plus symbol (`+`) as a *continuation prompt* and await the user to complete the command:

```
> 1 + 2 +
+ 3
[1] 6
```

In the second line, the plus symbol was produced by R while the value `3` was subsequently typed by the user. This can be confusing. It is possible to customize the continuation prompt (and numerous other aspects of R). For example, to indicate continuation by indenting with two spaces:

```
> options("continue" = "  ")
> 1 + 2 +
  3
[1] 6
```

Sometimes a mistake is discovered while at the continuation prompt. To abort the command and return to the command prompt, press esc.

1.2 The R Editor

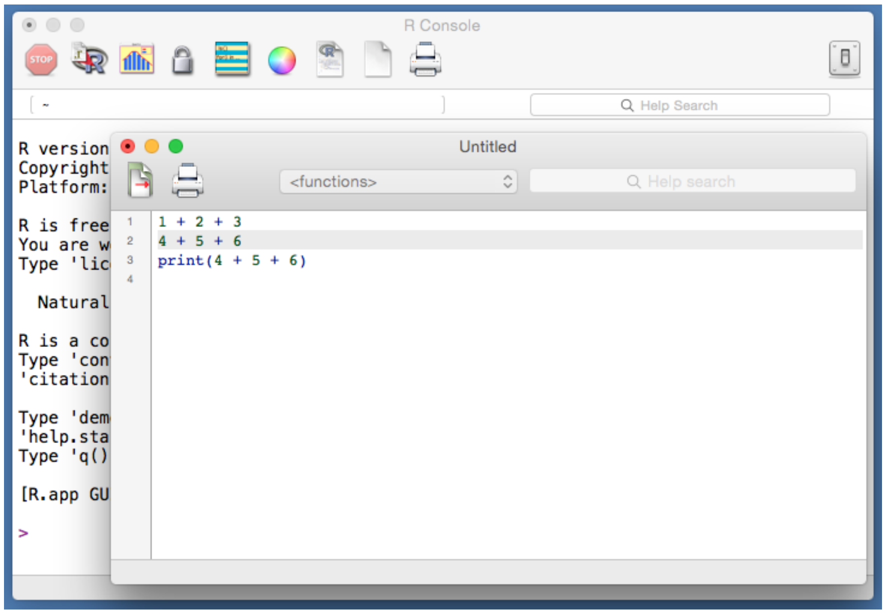
It is easy to mistype a command, especially when it is complex and spans multiple lines. Although R provides command-line editing, such as using ↑ and ↓ to scroll through previous commands, a better approach is to write a script or a list of commands in an editor. Besides making it easier to find and fix errors, a script can be saved for future reuse.

R provides an integrated editor which allows a script to be executed in part or whole from within the editor. In Figure 1.1, notice a menu bar just above the console window;[1] selecting File › New Document opens an editor window. Alternatively, one can use a keyboard shortcut, a method preferred by experienced users since it is quicker than mouse navigation. A keyboard shortcut comprises a *modifier key* and a character key. A modifier key is a special key that temporarily alters the action of other keys or mouse clicks. For example, pressing A normally produces a lowercase "a" but pressing shift+A produces an uppercase "A". To execute a keyboard shortcut, the modifier key is held down while the character key is pressed. For example, to open an editor window, press ⌘+N on OS X or Ctrl+N on Windows.

Figure 1.2 shows an R editor window, with three lines of commands, atop the console window. To execute a single line of command, such as the second line "`4 + 5 + 6`", place the cursor anywhere on the line (which would automatically highlight the line) and press ⌘+return on OS X or Ctrl+R on Windows. The command will appear in the console window along with any result. To execute multiple lines of commands, highlight the lines (such as by dragging the mouse across the lines) and press the aforementioned keyboard shortcut to execute commands.

There are two ways to execute all the commands in an R script. One is to highlight the entire content by pressing ⌘+A on OS X or Ctrl+A on

[1]On OS X, the menu bar actually appears at the top of the screen.



```
R version
Copyright
Platform:

R is free
You are w
Type 'lic

  Natural

R is a co
Type 'con
'citation

Type 'dem
'help.sta
Type 'q()

[R.app GU

>
```

```
1 + 2 + 3
4 + 5 + 6
print(4 + 5 + 6)
```

Figure 1.2: The R editor on Mac OS X.

Windows, followed by the keyboard shortcut for executing a command. The other assumes that the script has been saved.[2] From the menu bar, select File ❯ Source File... and then locate and double-click on the name of the script in the dialog box. This process of sending commands to R for execution is referred to as "sourcing a script." Note that R responds to a sourced script a little differently; specifically, the results are not printed unless the user provides instructions to do so. The simplest way to print a result is to use the `print(`*`something`*`)` command, as shown on the third line of the script in Figure 1.2.

An R script can include *comments*, which are explanatory notes that are not meant to be evaluated by R. While a script may be written for personal use, there often comes a time when modification is necessary and, without comments, important details about how the code worked could have been forgotten by then. In R, any text that follows the symbol "`#`" is ignored, whether in a script or at the command prompt:

```
> 1 + 2 # This is ignored by R.
[1] 3
```

[2]By convention, the name of an R script ends with the suffix "`.R`".

```
> 1 + 2   This is not.
Error: unexpected symbol in "1 + 2   This"
```

1.3 The Working Directory

Each R session has associated with it a *working directory* (or folder) where files are retrieved from or saved to by default. Understanding this concept can help avoid many frustrations, such as searching every folder imaginable on the computer for a previously saved file or sourcing an outdated R script from an unintended folder. The command `getwd()` gives the working directory, which is usually `/Users/`*`username`* (not literally *`username`*) on OS X and `C:\Users\`*`username`*`\Documents` on Windows.

As an illustration, consider sourcing from the R console a script previously saved as “`myScript.R`”. This is done using the `source("`*`filename`*`")` command:

```
> source("myScript.R")
```

If “`myScript.R`” is found in the working directory, R will execute its content; otherwise, an error message will be printed, the last two lines of which are:

```
In file(filename, "r", encoding = encoding) :
  cannot open file 'myScript.R': No such file or directory
```

To understand R’s response, imagine looking for someone by the name of James Smith in a large organization with many offices in a building. If you stumble into one of the offices and ask for James Smith, chances are you will get a response similar to the above (some polite variant of “*cannot find James Smith: No such person*”) or be greeted by a wrong James Smith. A better approach to look for James Smith would be to specify, in addition to his name, the division and department he works for. This information can be represented hierarchically as `/Division/Department/James Smith`.

Files on a computer are grouped into folders, which are organized in a hierarchy. The *absolute pathname* of a file describes its location in the hierarchy by tracing a path from the top-most folder, called the *root directory*, through all the intermediate folders, to the file. It begins with a “`/`” on OS X and “`C:\`” on Windows, where the letter “`C`” could be replaced by some other letter that identifies a drive or partition on the computer. This is illustrated in Figure 1.3 for OS X. The absolute pathname

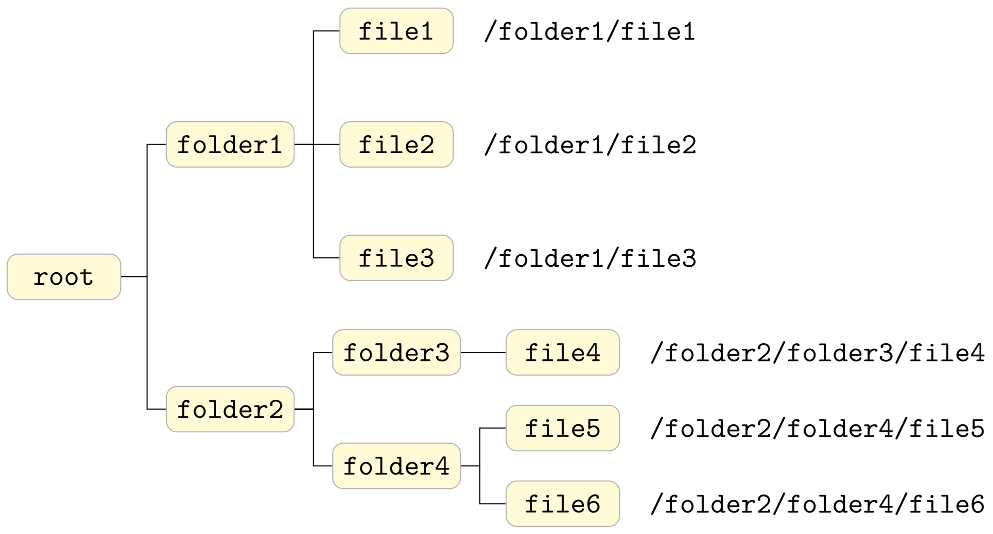


Figure 1.3: The hierarchical file structure and absolute pathnames.

"`/folder2/folder4/file6`" in the bottom right corner indicates that `file6` is a file located in a folder named `folder4`, which in turn is located in a folder named `folder2` in the root directory. The symbol "`/`" thus distinguishes folder levels, with the first "`/`" representing the root directory. If the name of a file is specified without a "`/`" on OS X or "`C:\`" on Windows, say, simply as "`myScript.R`", it is a *relative pathname* and the file is assumed to be located in the working directory. In other words, R assumes that the absolute pathname of the file is "`/Users/`*`username`*`/myScript.R`" on OS X or "`C:\Users\`*`username`*`\Documents\myScript.R`" on Windows.

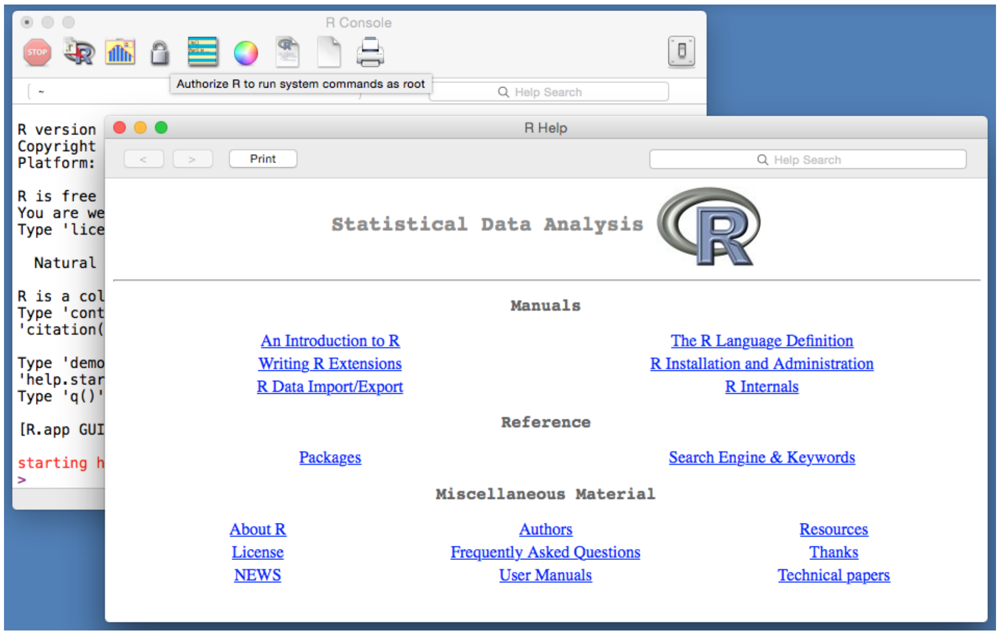
It is useful to organize projects into directories and, when working on a particular project, set R's working directory to the associated directory. This can be done using the command `setwd("`*`absolutePathName`*`")`. For example, on the author's computer, the default working directory is:

```
> getwd()
[1] "/Users/blee"
```

A folder named `book` has been created in `/Users/blee` to store files related to this book. To designate this folder as the working directory:

```
> setwd("/Users/blee/book")
```

On Windows, the equivalent command would be:



```
R version
Copyright
Platform:

R is free
You are we
Type 'lice

  Natural

R is a col
Type 'cont
'citation(

Type 'demo
'help.star
Type 'q()'

[R.app GUI

starting h
>
```

Statistical Data Analysis

Manuals

An Introduction to R
The R Language Definition
Writing R Extensions
R Installation and Administration
R Data Import/Export
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
Authors
Resources
License
Frequently Asked Questions
Thanks
NEWS
User Manuals
Technical papers

Figure 1.4: The R help window on Mac OS X.

```
> setwd("C:/Users/blee/Documents/book")
```

Note that pathnames are always specified with "/" in R, even on Windows; using "\" is a common mistake among beginning users who are accustomed to Windows.

1.4 Getting Help

R has a comprehensive built-in help system which may be accessed in several ways, one of which is to select Help ⟩ R Help from the menu bar. This presents a window similar to that shown in Figure 1.4, wherein users can search for help on various commands and find links to several manuals, in particular the highly recommended "An Introduction to R" by Venables, Smith and the R Core Team [13]. The `help(`*`topic`*`)` command can alternatively be used to invoke the help system. For example, to display the documentation on the `print` command, either type `help(print)` or `?print` at the command prompt, where the question mark ("`?`") is a shortcut for the command `help`.

There is also a plethora of internet resources for learning R and getting help with problems. A few of these are listed below.

- CRAN, which contains the links shown in Figure 1.4 and more.
- R-Bloggers (`http://www.r-bloggers.com/`), where daily news and tutorials about R can be found.
- Rseek (`http://rseek.org/`), an R-specific search engine.
- Stack Overflow (`http://stackoverflow.com/tags/r`), a site with a very active R community asking and answering questions about R.

1.5 Exercises

1. By default, a short introductory message is displayed in the console window when R starts.

   (a) Write down the version number of R installed on your computer.
   (b) Write down the platform under which R is running on your computer.
   (c) List the four commands displayed in the console window just above the first command prompt.

2. The keyboard shortcuts for some commands can be found in the drop-down menus in the menu bar (usually to the right of the listed commands). For example, one way to save an R script is to select File > Save from the menu bar; the corresponding keyboard shortcut is ⌘+S on OS X or Ctrl+S on Windows. List the keyboard shortcuts for the following.

   (a) File > Open Document on OS X or File > Open script on Windows, which opens a dialog box to load a previously saved script into the R editor.
   (b) Edit > Clear Console, which clears the screen in the console window.

3. Create an R script that contains the following two lines:

   ```
   1 - 2 + 3 * 4 / 6
   print(1 - 2 + 3 * 4 / 6)
   ```

   Create a folder named `compStat` in the default working directory of R and save the script as `ex1-3.R` in the folder.

   (a) In the R editor, execute the first line of `ex1-3.R` and note the result displayed in the R console.

(b) Source `ex1-3.R` from the R console and note the result displayed. Which line of command does the result correspond to? Explain.

Chapter 2

Fundamental Objects

Everything that exists in R is an object. This includes constants, data sets, functions and graphs. But what is an object in R? Put simply, it is a container for information referred to as *value* and it is self-describing—like a labeled food jar, except that it usually has a name. Among the descriptions attached to an object is its *class*, which defines what it contains as well as the way its content is organized. This basic concept of an object is one of the keys to understanding how to work with data in R, for a common mistake in R is to attempt to perform an operation on an object of a class that is incompatible with the operation. In this chapter, we present the rudiments of working with some of the fundamental objects in R, including assignments, special values, environments, functions, atomic vectors and lists.

2.1 Expressions and Assignment

Any command that is typed into the console is an *expression*, a symbol or a combination of symbols that evaluates to a value. For example, `1 + 2` is an expression which evaluates to the value 3. When R evaluates an expression, an object is created somewhere in the computer memory to store the value of the expression. The object (and hence the value it contains) is only accessible by name; an anonymous object gets deleted from the computer memory by a process called *garbage collection*. Thus, most objects are created by an operation called *assignment*, which establishes a *binding* or an association between an object and a name. This is usually done with the use of a symbol composed of a less-than sign followed by a minus sign ("`<-`"), pronounced as "gets". For example:

```
> x <- 1 + 2 - 3
```

Here an object is created when R first evaluates the expression "`1 + 2 - 3`". The object holds the value of the expression and is bound to the name `x` by assignment. It is now accessible by name. For example, typing its name into the console displays its value (to the screen by default):

```
> x
[1] 0
```

It can also be used to create further objects:

```
> y <- x - 2
> y
[1] -2
```

There is a special object named `.Last.value` that contains the value of the last evaluated expression:

```
> 1 + 2 - 3
[1] 0
> .Last.value
[1] 0
> 3 + 2 - 1
[1] 4
> .Last.value
[1] 4
```

It might come in handy when a computationally intensive expression has been evaluated but, by an oversight, its value has not been assigned a name.

An object name can contain letters, digits, periods ("`.`") and underscores ("`_`"), with the restriction that it must begin with a letter or a period. If it begins with a period, it cannot be followed by a digit. R is case sensitive, for example, it considers `y` and `Y` to refer to different objects. It is useful to keep in mind that the name of an object is just a symbolic representation of the location of the object in the computer memory, analogous to a postal address that is used to indicate the location of a building. In other words, an object and its name are separate entities.[1] In the preceding listings, for

[1]In fact, the name of an object itself is an object, one which a user rarely needs to deal with and will not be discussed here.

example, any subsequent changes to the value of `x` does not affect the value of `y`:

```
> x <- 100
> y
[1] -2
```

It is the object that contains the value of `x - 2`, at the time of assignment, that is bound to `y`.

Assignments can alternatively be made using the symbol "`=`":

```
> x = 1 + 2 - 3
```

Some may prefer the symbol "`=`" to the symbol "`<-`" because it requires one less keystroke, but the two are not equivalent: "`<-`" always represents assignment whereas "`=`" can represent assignment (and more).

2.2 Special Values

Certain words in R are reserved and may not be used to name an object. Among these are *special values* that represent mathematical abstractions, rather than values in the usual sense, described as follows.

`Inf` Most students of mathematics recognize the symbol "∞", called the lemniscate, which represents the concept of infinity. While infinity can be an elusive mathematical concept,[2] it means one of two things in R: either the value of an expression is too large (positively or negatively) to be represented by the computer, or a nonzero value is being divided by zero. These are indicated by the special value `Inf`. For example:

```
> 10^400
[1] Inf
> 1 - 10^400
[1] -Inf
> 1 / 0
[1] Inf
```

[2]A persistant myth in the history of mathematics is that Georg Cantor, a pre-eminent mathematician, descended into isolation and insanity as a consequence of his inability to resolve important questions about infinity.

Every computer has a well-defined range of values that it can represent; for most, the largest number allowed is approximately 1.79×10^{308}:

```
> 1.79 * 10^308
[1] 1.79e+308
> 1.8 * 10^308
[1] Inf
```

Note that R uses the notation `aeb`, where `a` and `b` are numbers, to stand for $\mathtt{a} \times 10^{\mathtt{b}}$; that is, the output `1.79e+308` means 1.79×10^{308}. This notation can also be used at the command prompt.

`NaN` When the result of a computation is undefined, R returns the special value `NaN`, which stands for "not a number." For example:

```
> 0 / 0
[1] NaN
> Inf / Inf
[1] NaN
> sqrt(-1)
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
```

Notice that a warning message is printed only when the expression "`sqrt(-1)`" is evaluated. The reason is, unlike $0/0$ or ∞/∞, both of which are mathematically undefined, $\sqrt{-1}$ is defined in the complex number system, which is understood by R if the number -1 is explicitly expressed as a complex number (see page 36).

`NA` Missing values are a common occurrence in real-world data sets, which arise because some values are corrupted or unobserved to begin with. In R, missing values are represented with the special value `NA`, which stands for "not available." In general, any computations involving an `NA` results in an `NA`:

```
> NA - NA
[1] NA
> sqrt(NA)
[1] NA
> NA / 0
[1] NA
```

`NULL` refers to a special object in R that is used to indicate that an object does not exist. It is often confused with the special value `NA`. To illustrate the difference between `NULL` and `NA`, the command `length(`*`object`*`)` can be used to find out the number of elements each object contains:

```
> length(NA)
[1] 1
> length(NULL)
[1] 0
```

Note that `NULL` is not the only object in R that contains no elements. It is useful to think of `NA` as a placeholder for a value that should have been there, whereas `NULL` is a nonexistent value.

2.3 Classes

The class of an object describes its content, so that the object can be handled in an appropriate manner. Specifically, the same command, when applied to objects of different classes, can lead to different operations and produce different results. For example, consider the addition of two values using "`+`". If both values are ordinary numbers, "`+`" is just a basic arithmetic operation:

```
> x <- 25
> x + 7
[1] 32
```

If one of the values is a date and the other is a number, a different operation is performed:

```
> y <- Sys.Date()
> y
[1] "2015-05-25"
> y + 7
[1] "2015-06-01"
```

Here the command `Sys.Date()` returns an object that contains the current date, which is bound to the name `y`. When the value `7` is added to the current date, the result is the date seven days later. How does R know that a different operation is called for? Because R first queries an object for its class, then selects an operation that is appropriate for that class of objects, if available. The command `class(`*`object`*`)` returns the class of an object:

```
> class(x)
[1] "numeric"
> class(y)
[1] "Date"
```

Several classes of objects will be introduced in Section 2.6 and 2.7.

2.4 Environments

When R evaluates an expression that contains a name, say, "`x + 1`", how does R locate the object associated with the name "`x`" to retrieve the value? Effective use of R for data analysis requires an understanding of how R organizes objects created within it. The concept is a familiar one, as will be seen shortly.

During an assignment, the binding (or association) between an object and its name is stored in a specialized object called *environment*. More than one environment exists during an R session; in fact, in a typical R session, environments are constantly being created and destroyed, as discussed in the next section (page 30). In addition to a set of bindings, an environment contains a pointer to another environment, called the *enclosing* or *parent* environment. This is illustrated in Figure 2.1. It follows that there is a chain of environments, like a hierarchy of folders used to store files on a computer, where each folder contains a single file that catalogs a set of names and the locations of the associated objects, and a folder with a similar content. The chain of environments ends with the *empty* environment, the only environment without an enclosing environment and, as the name suggests, contains no bindings.

Whenever an expression is evaluated, one of the environments is "active", referred to as the *local* environment. To resolve a name that appears in the expression, R searches the local environment for a name that matches. If a matching name is found, the associated object is located and returned; otherwise, the enclosing environment is examined and the process repeated. An error is signalled if all the enclosing environments have been searched and a matching name cannot be found:

```
> x + 1
Error: object 'x' not found
```

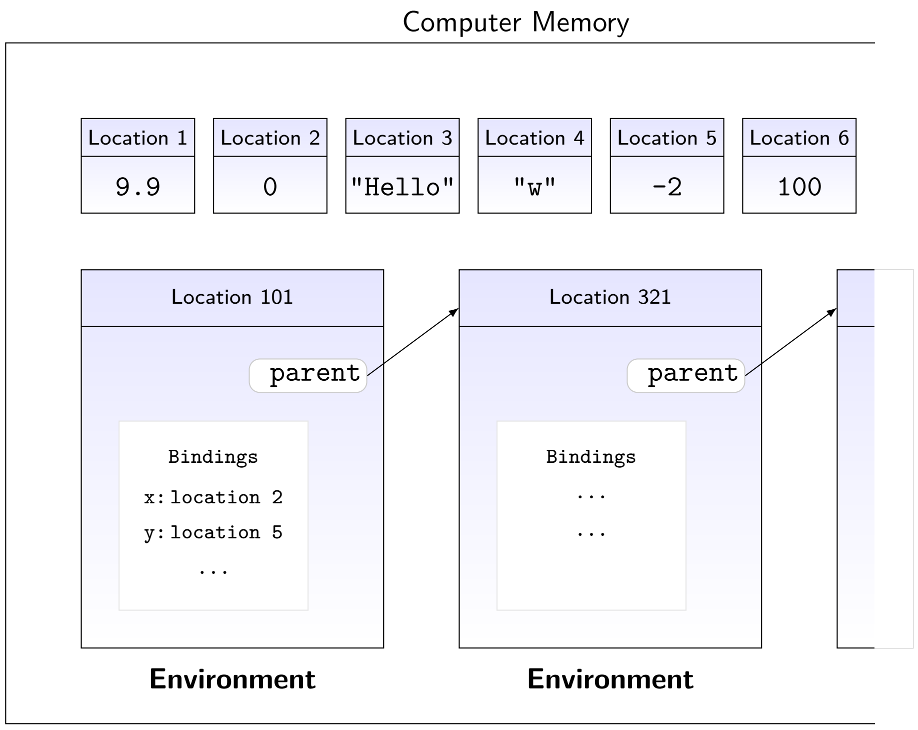


Figure 2.1: A schematic representation of an environment and its enclosing environment.

An analogy would be a "full-service" library, where a patron simply walks to the front desk to ask for a book by name. The librarian first searches the library catalog for a matching name to locate the shelf that holds the book. If found, the book is retrieved and handed to the patron; otherwise, the librarian proceeds to search the catalog of a branch library, until all branch libraries are exhausted.

Most users do not need to explicitly deal with environments. For now, it suffices to know that when R starts, a *global* environment or *workspace* is created, and every subsequent assignment that takes place at the command prompt adds an entry to it. In other words, the workspace is the local environment of all assignments that take place at the command prompt. The command `ls()` may be used to display the names contained in the workspace, which is usually empty at the beginning of an R session:

```
> ls()
```

```
character(0)
```

Here "`character(0)`" may be interpreted as "empty" for now. New assignments create new bindings that populate the workspace:

```
> x <- 1
> y <- 2
> z <- 3
> ls()
[1] "x" "y" "z"
```

To remove one or more bindings, use the command `rm(`*`objects`*`)`:

```
> rm(x)
> ls()
[1] "y" "z"
> rm(y, z)
> ls()
character(0)
```

When R evaluates the command "`rm(x)`", for example, it dissociates the name `x` and the object that was originally bound to it. The name `x` is removed from the workspace, like erasing an entry in a catalog, but nothing is done to the object itself.

2.5 Functions

A *function* in R is similar to a function in mathematics, in that it may take one or more *arguments* as input, performs some operations and produces an output. The input corresponds to a set of objects and the output is another object. Unlike a mathematical function, however, an R function may require no arguments.

Calling Functions

R has many built-in functions, including the mathematical functions found on most scientific calculators, such as power, exponential and logarithmic. A function *call* is a command to execute the code of a function, like pressing one or more buttons on a calculator to perform a calculation. It usually