01

CHAPTER

基本概念

1-1 Big Data 的起源

Big Data 一词，有人解释为大数据，也有人解释为巨量资料，其实都 OK，本书则以大数据为主要用法。

2012 年世界经济论坛在瑞士达沃夫（Davos）有一个主要议题“Big Data, Big Impact”，同年《纽约时报》（The New York Times，如右图所示）的一篇文章，《How Big Data Became So Big》，清楚揭露大数据时代已经降临，它可以用在商业、经济和其他领域中。

本图片取材自 The New York Times

1-2 R 语言之美

大数据需处理的数据是广泛的，基本上可分成两大类，有序数据与无序数据，对于有序数据，目前许多程序语言已可处理。但对于无序数据，例如，地理位置信息、Facebook 信息、视频数据等，是无法处理的。而 R 语言正可以解决这方面的问题，自此 R 已成为有志成为信息科学家（Data Scientist）或大数据工程师（Big Data Engineer）所必需精通的计算机语言。

Google 首席经济学家 Hal Ronald Varian，如右图所示，有一句经典名言形容 R。

The Great beauty of R is that you can modify it to do all sorts of things. And you have a lot of prepackaged stuff that's already available, so you're standing on the shoulders of giants.

大意是，R 语言之美在于，你可以通过修改很多高手已经写好的套件程序，解决各式各样的问题。因此，当你使用 R 语言时，你已经站在巨人的肩膀上了。

本图片取材自 Wikipedia

1-3 R 语言的起源

提到 R 语言，不得不提 John Chambers，如下图所示。他是加拿大多伦多大学毕业，然后拿到哈佛大学统计硕士和博士。

John Chambers 在 1976 年于 Bell 实验室工作时，为了节省使用 SAS 和 SPSS 软件经费，以 Fortran 为基础，开发了 S 语言。这个 S 语言主要是处理，向量（Vector）、矩阵（Matrix）、数组（Array）以及进行图表和统计分析的，初期只是可以在 Bell 实验室的系统上运行，随后这个 S 语言被移植至早期的 Unix 系统下运行。然后 Bell 实验室以很低的廉价格授权各大学使用。

John Chambers 本图片取材自网络

R 语言主要是以 S 语言为基础，开发完成。

1993 年新西兰 University of Auckland 大学统计系的教授 Ross Ihaka 和 Robert Gentleman 两位 R 先生，分别如下图（左）和下图（右）所示，为了方便教授统计学，以 S 语言为基础开发完成一个程序语言，因为他两人名前缀字皆是 R，于是他们所开发的语言就被称为 R 语言，其 Logo 如下图（右）所示。

Ross Ihaka

本图片取材自网络

Robert Gentleman

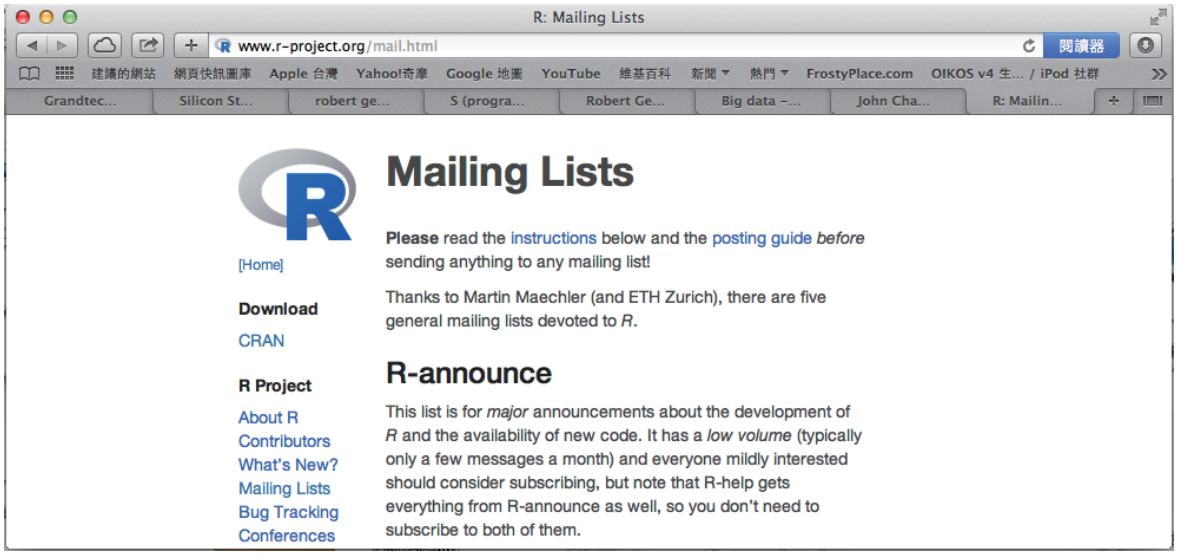
本图片取材自网络

R 语言标准 Logo

现在的 R 语言则由一个 R 核心开发团队负责，当然 Ross Ihaka 和 Robert Gentleman 是这个开发团队的成员，另外，S 语言的开发者 John Chambers 也是这个 R 语言开发团队的成员。目前这个开发团队共有 18 名成员，这些成员拥有修改 R 核心代码的权限。下列是 R 语言开发的几个有意义的时间点。

1） 1990 年代初期 R 语言被开发。

2） 1993 年 Ross Ihaka 和 Robert Gentleman 开发了 R 语言软件，在 S-news 邮件中发表。吸引了一些人关注并和他们合作，自此一组针对 R 的邮件被建立。如果你想了解更多这方面的信息可参考下图中的网址。

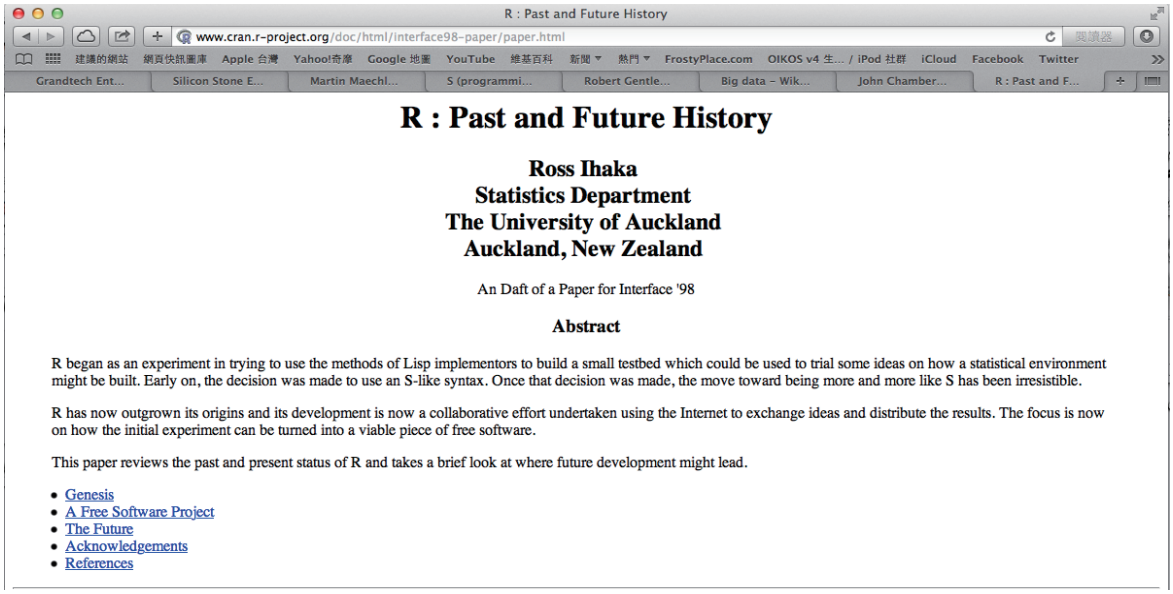


3） 1995 年 6 月在 Martin Maechler（如下图所示）等人的努力下，这个 R 语言被同意免费使用，同时遵守自由软件基金会（Free Software Foundation）的 GNU General Public License（GNU 通用公共许可证，GPL）Version 2 的协议。

Dr. Martin Maechler 取材自 stat.ethz.ch/people/maechler

4） 1997 年 R 语言核心开发团队成立。

5） 2000 年第 1 版 R1.0.0 正式发布。Ross Ihaka 将 R 的开发简史记录了下来，可参考下图中的网址。


R : Past and Future History

www.cran.r-project.org/doc/html/interface98-paper/paper.html

R : Past and Future History

Ross Ihaka
Statistics Department
The University of Auckland
Auckland, New Zealand

An Daft of a Paper for Interface '98

Abstract

R began as an experiment in trying to use the methods of Lisp implementors to build a small testbed which could be used to trial some ideas on how a statistical environment might be built. Early on, the decision was made to use an S-like syntax. Once that decision was made, the move toward being more and more like S has been irresistible.

R has now outgrown its origins and its development is now a collaborative effort undertaken using the Internet to exchange ideas and distribute the results. The focus is now on how the initial experiment can be turned into a viable piece of free software.

This paper reviews the past and present status of R and takes a brief look at where future development might lead.

1-4 R 的运行环境

在 R 语言核心开发团队的努力下，目前 R 语言已可以在常见的各种操作系统下运行。例如，Windows、Mac OS、Unix 和 Linux。

1-5 R 的扩展

R 的一个重要优点是，R 是 Open Source License，这表示任何人均可下载并修改，因此许多人在编写增强功能的套件，同时供他人免费使用。

1-6 本书的学习目标

不容否认，不论是 S 语言或 R 语言均是统计专家所开发的，因此，R 具有可以完成各种统计的工具。但已有越来越多的程序设计师开始加入学习 R，使得 R 也开始可以完成非统计方面的工作，例如，数据处理、图形处理、心理学、遗传学、生物学、市场调查等等。

本书在编写时，尽量将读者视为初学者，辅以丰富实例，期待读者可以用最轻松的方式学会 R 语言。

本章习题

一、判断题

（　　）1. 要成为大数据工程师（Big Data Engineer），学习 R 语言是一件很重要的事。

（　　）2.Facebook 信息、视频数据是可排序的数据。

（　　）3.R 语言目前只能在 Windows 和 Mac OS 系统下执行。

（　　）4.R 语言是免费软件。

二、单选题

（　　）1. R 语言无法在以下哪一个系统下执行？

A. Linux　　B. Unix　　C. Android　　D. Mac OS

（　　）2. 下列哪一个人对 R 语言的开发没有贡献？

A. Steve Job　　B. Ross Ihaka

C. John Chambers　　D. Robert Gentleman

（　　）3. R 语言是以哪一个语言为基础开发完成？

A. SAS　　B. S　　C. SPSS　　D. C

三、多选题

（　　）1. 我们现在可以免费使用 R 语言，下列哪些人是有贡献的？（选择 3 项）

A. Martin Maechler　　B. Ross Ihaka

C. Robert Gentleman　　D. Tim Cook

E. Marissa Mayer

02

CHAPTER

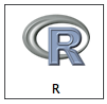
第一次使用 R

有关安装 R 语言程序与 RStudio 作业环境套件的操作可以参考附录 A，本章笔者将介绍如何启动和在 R Console 窗口下撰写 R 程序。

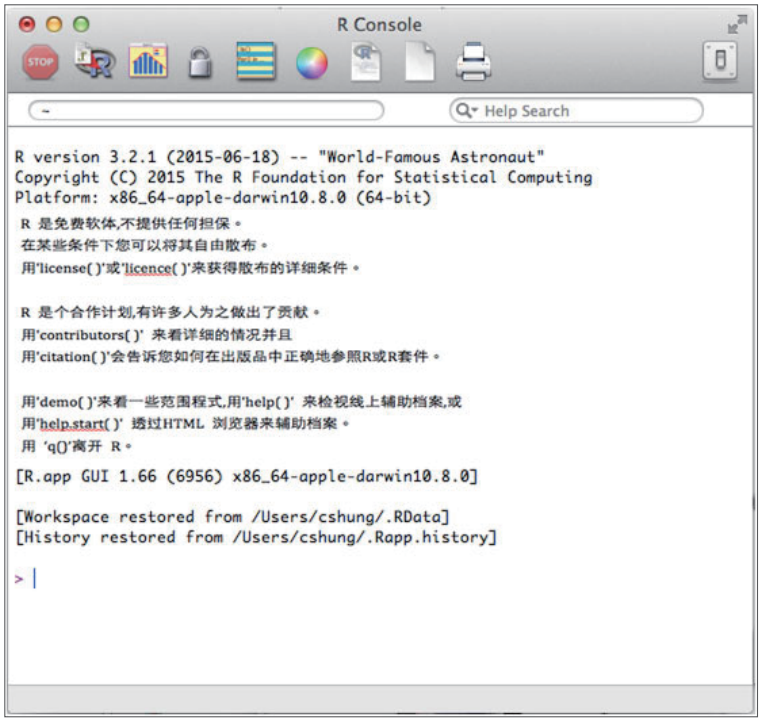
2-1 第一次启动 R

2-1-1 在 Mac OS 下启动 R

在 Mac 环境中，如果先前只是安装 R，并没有安装 RStudio，则可以在应用程序文件夹看到 R 语言图标，如下图所示，然后启动。



双击标准 R 图标，可以正式进入 R-Console 环境，如下图所示。

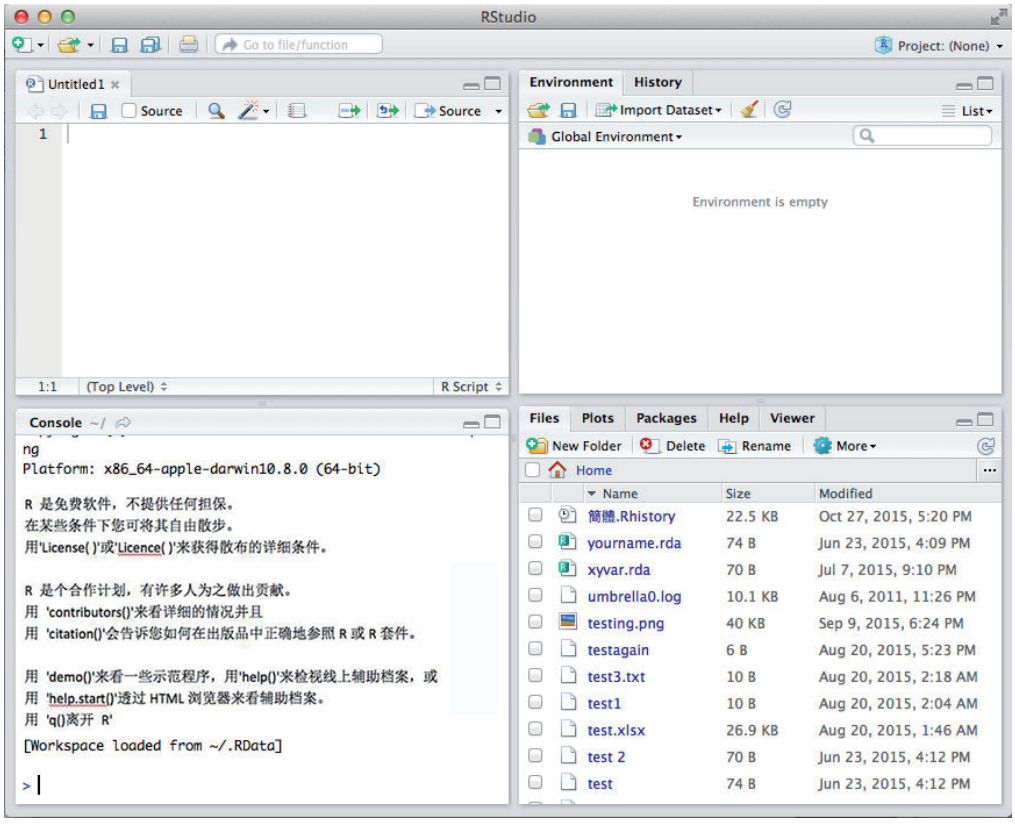


在这里，就可以正式使用 R 语言了。

2-1-2 在 Mac OS 下启动 RStudio

如果你安装完 R，然后也安装完 RStudio，则可以在屏幕下方工具栏看到 RStudio 图标，如下图所示。

即可以启动，R 的整合式窗口环境如下图所示。

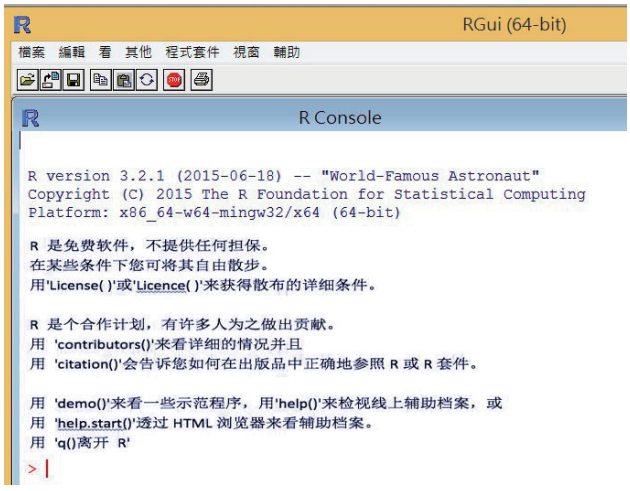


由上图可以看到整合式窗口共有 4 个区域，基本上左下方的 Console 窗口，是我们最常使用的窗口。

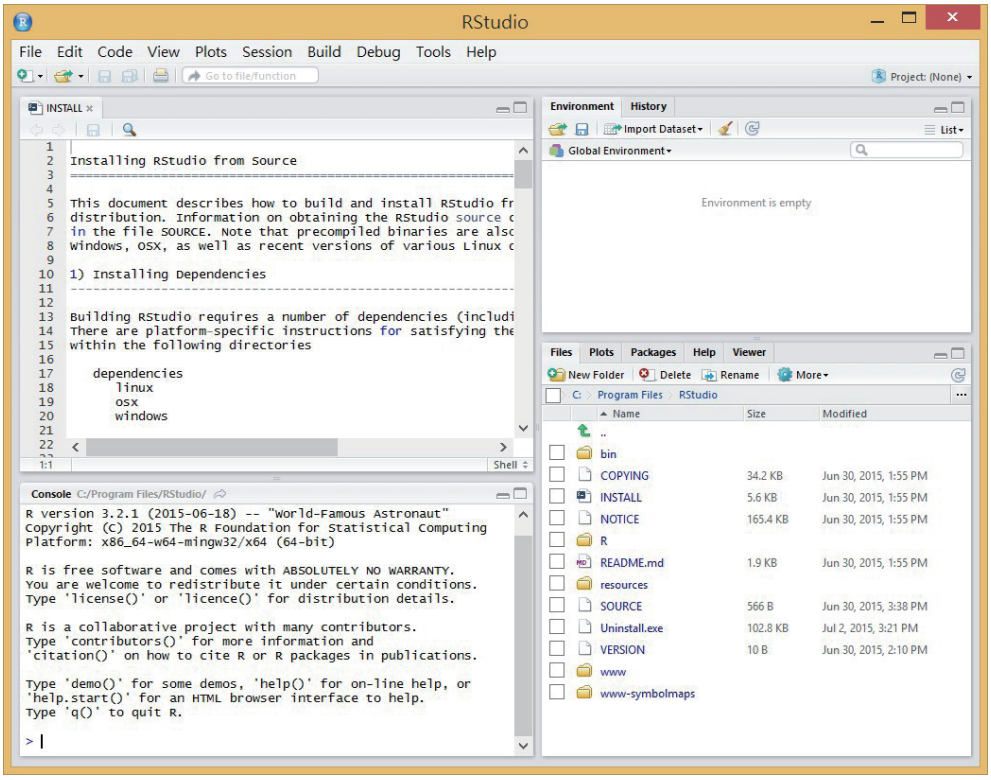
未来所有实例，均是在 RStudio 窗口内执行的。

2–1–3 在 Windows 环境中启动 R 和 RStudio

安装完成 Windows 系统的 R 后，如果启动 R，可以看到下列 R-Console 窗口。



如果安装并启动 RStudio，就可以看到下列 RStudio 窗口。



2-2 认识 RStudio 环境

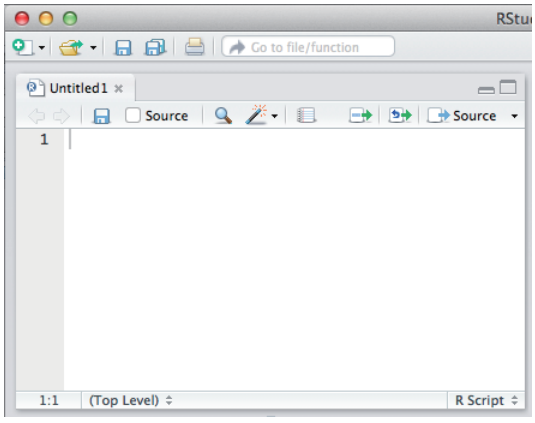
可参考下图，基本上可以将 RStudio 整合式窗口分成以下 4 大区域。

1. Source Editor

Soure Editor 区域位于 RStudio 窗口左上角，如下图（左）所示。这是 R 语言的程序代码编辑区，你可以在此编辑 R 语言程序代码，储存，最后再运行。

2. Console

Console 区域位于 RStudio 窗口左下角，如下图（右）所示。R 语言也可以支持直译器（Interpreter）功能，此时就需要使用此区域窗口，在此可以直接输入指令，同时获得执行结果。



3. Workspace

Workspace 窗口位于 RStudio 窗口右上角，如下图（左）所示。如果选择 Environment 标签，那么这区域会记录在 Console 输入的所有指令的相关对象的变量名称和值。如果选择 History 标签，则可以在此看到 Console 窗口所有执行指令的记录。

4. Files、Plots、Packages、Help 和 Viewer

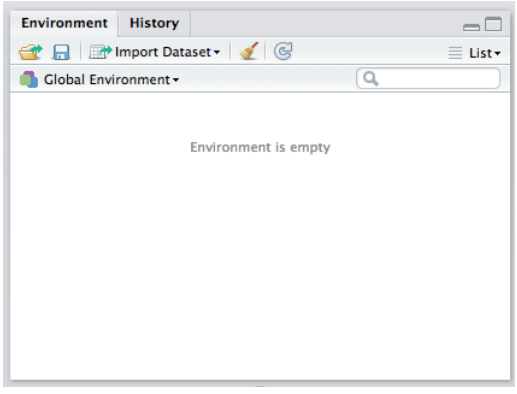
该区域位于 RStudio 窗口右下角，如下图（右）所示。这几个标签的功能分别如下所述。

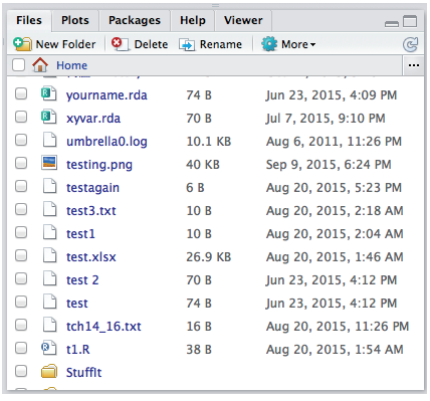
1）Files：在此可以查看个文件夹的内容。

2）Plots：在此可以呈现图表。

3）Packages：在此可以看到已安装 R 的扩充套件。

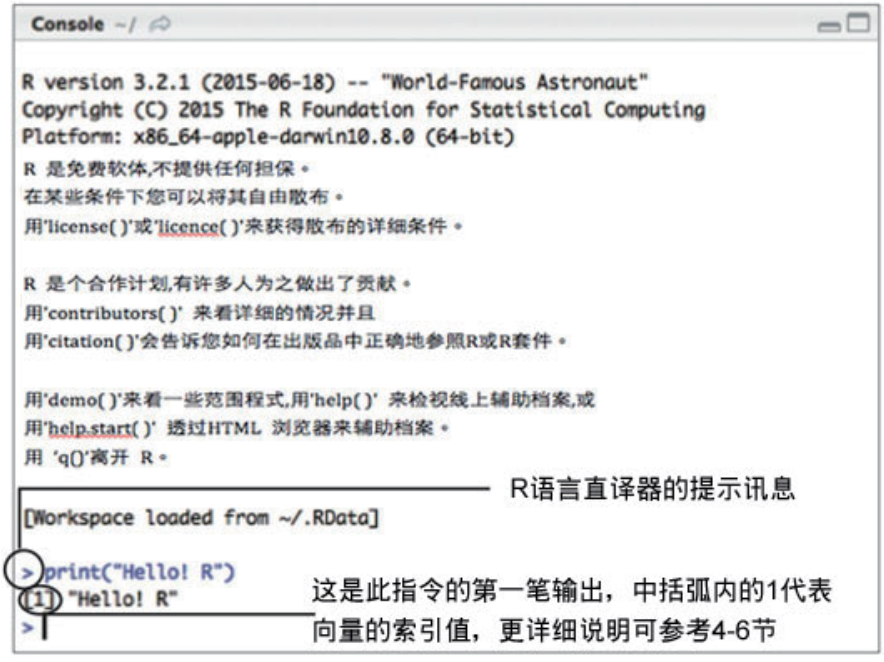
4）Help：在此可浏览辅助说明文件内容。





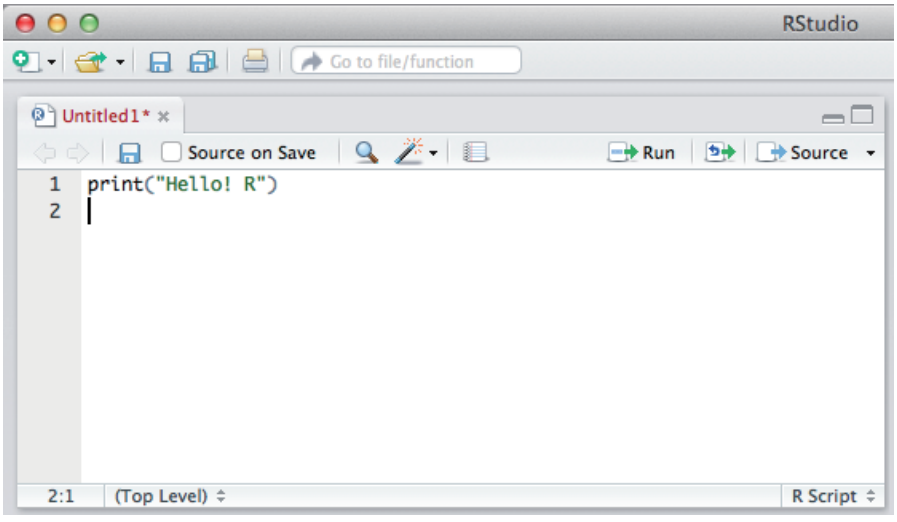
2-3 第一次使用 R

先前说过 R 可以支持直译器功能，下图所示的是打印“Hello! R”，可参考下图所示的使用 Console 窗口的操作范例和结果。

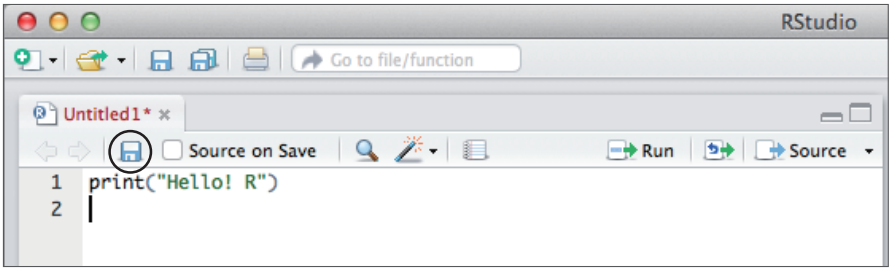


在上图中可以了解到，“>”是 R 语言直译器的提示信息，当看到此信息时，即可以输入 R 命令。

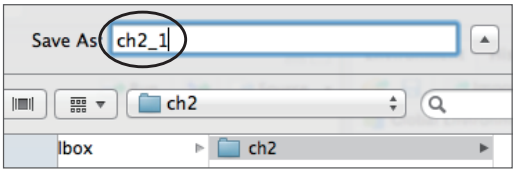
当然我们也可以使用 Source Editor 编辑程序，然后再执行。执行结果的实例，可参考下图。首先编辑下图所法的程序代码。



接着储存上述程序代码，如下图所示。

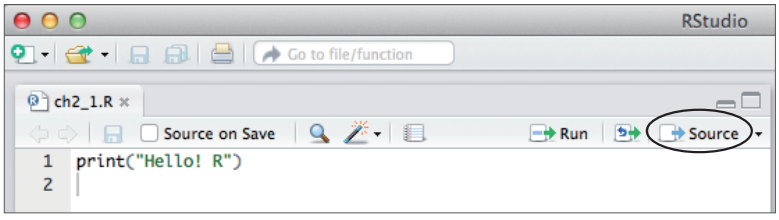


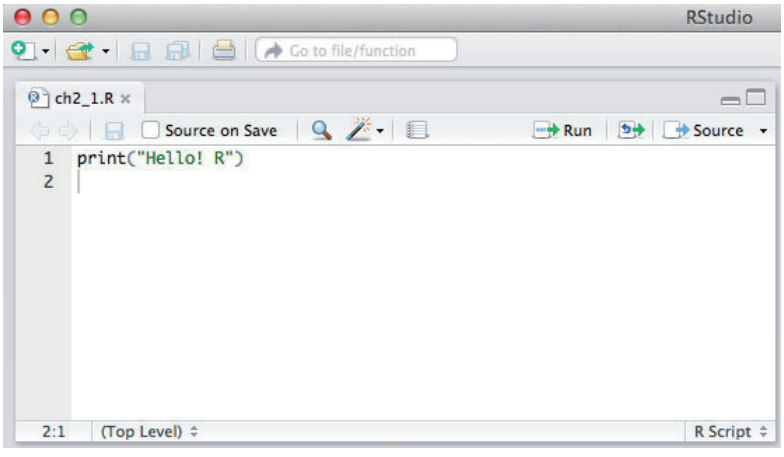
请单击上图中的储存按钮，也可以执行 RStudio 的 File/Save As 命令，接着选择适当的文件夹，再输入适当的文件名。此例的命件名是 ch2_1，如下图所示。R 语言默认的文件名扩展名是 R。



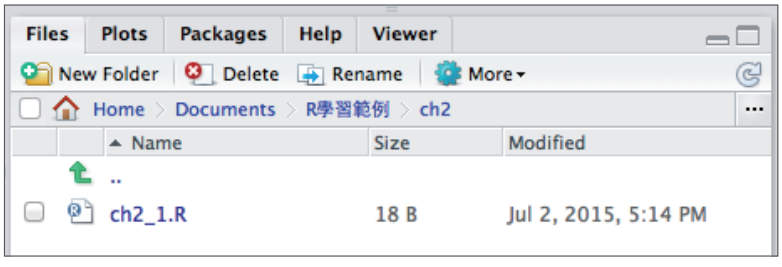
所以执行完上述命令，就相当于将代码储存在 ch2_1.R。

在 RStudio 的 Source Editor 区有“Source”标签，如下图所示。如果这时单击此标签，这个动作被称为 Sourcing a Script。其实这就是执行 Source Editor 工作区的程序（其实这个动作也会同时储存程序代码）。单击“Source”标签后可以看到下图所示的执行结果。

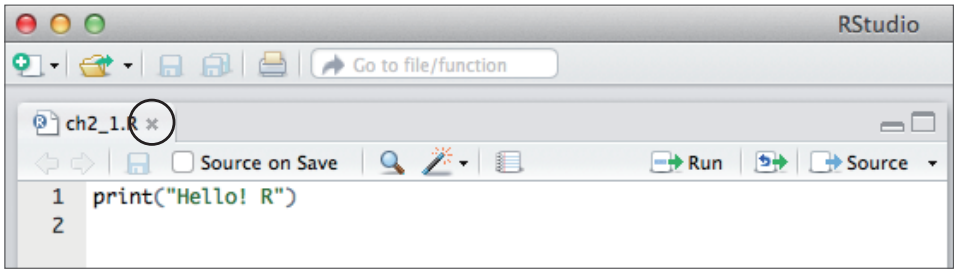




一个完整的 R 程序，即使是在 Source Editor 区编辑，其执行的非图形数据结果，也将是在 Console 窗口中显示，如上图所示。如果此时检查 RStudio 整合式窗口的右下方，再单击“Files”标签，适当地选择文件夹后，就可以看到 ch2_1.R 文件，如下图所示。



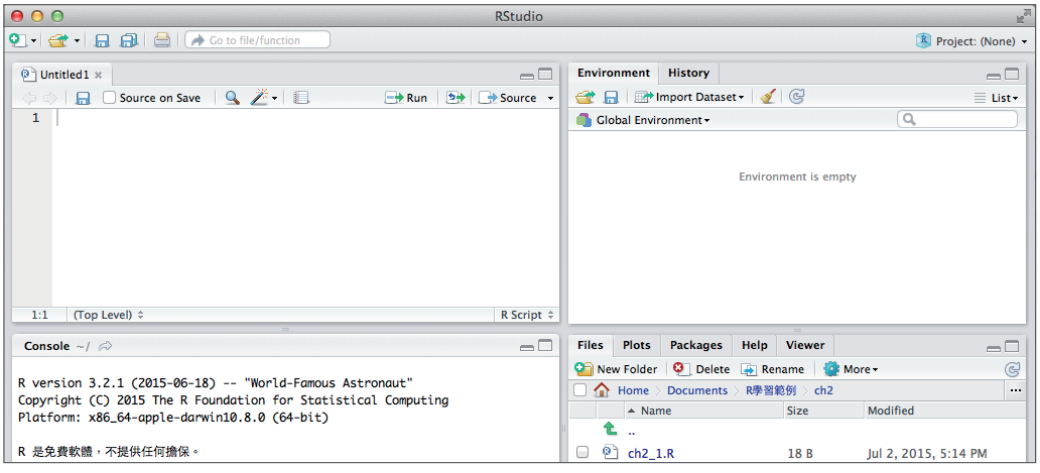
假设现在想编辑新的文件，可单击下图中“ch2_1.R”标签右边的关闭按钮。



此时 Source Editor 区的窗口会暂时消失。之后单击下图中 Console 窗口右上角的按钮。

```
Console ~/

R version 3.2.1 (2015-06-18) -- "World-Famous Astronaut"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R 是免费软件，不提供任何担保。
在某些条件下您可将其自由散步。
用'License( )'或'Licence( )'来获得散布的详细条件。

R 是个合作计划，有许多人为之做出贡献。
用 'contributors()'来看详细的情况并且
用 'citation()'会告诉您如何在出版品中正确地参照 R 或 R 套件。

用 'demo()'来看一些示范程序，用'help()'来检视线上辅助档案，或
用 'help.start()'透过 HTML 浏览器来看辅助档案。
用 'q()离开 R'

> print("Hello! R")
[1] "Hello! R"
> source('~/Documents/R学习范例 /ch2/ch2_1.R')
[1] "Hello! R"
>
```

便可恢复显示 Source Editor 窗口，如下图所示。



如果 Source Editor 窗口内，同时有多个文件被编辑时，关闭一个所编辑的文件，此时将改成显示其他编辑的文件。

2-4 R 语言的对象设定

如果你学过其他计算机语言，想将变量 x 设为 5，可使用下列方法。

x = 5

R 语言是一种面向对象的语言，上述 x，也可被称为对象变量。甚至，有的 R 程序设计师称 x 为对象。在本书本章中笔者先用完整名称“对象变量”，在后续章节中，笔者将直接以对象（Object）称之。

在 R 语言中，可以使用上述等号，但更多的 R 语言程序设计师，会使用“<-”符号，其实此符号与“=”号，意义一样。例如，将变量 x 设定为 5 可按如下方式。

x <- 5

可参考下列实例。

```
> x = 5
> x
[1] 5
> x <- 5
> x
[1] 5
>
```

在上述程序实例中，在给对象变量 x 赋值后，如果直接列出对象变量 x，则相当于可列出对象变量的值，此例是列出 5。至于“[1]”是指这是第一项输出。

另一个奇怪的 R 的等号表示方式，是以“->”表示，这种表示方式的对象变量是放在等号右边，如下所示。

5 -> x

可参考下列实例。

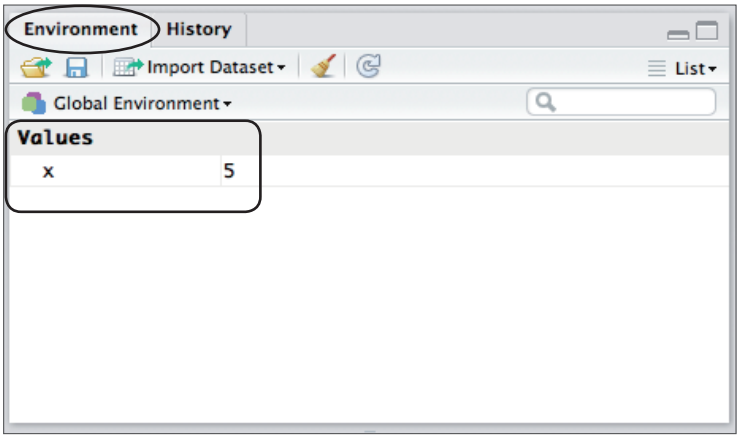
```
> 5 -> x
> x
[1] 5
>
```

不过这种方法，R 程序设计师一般比较少用。

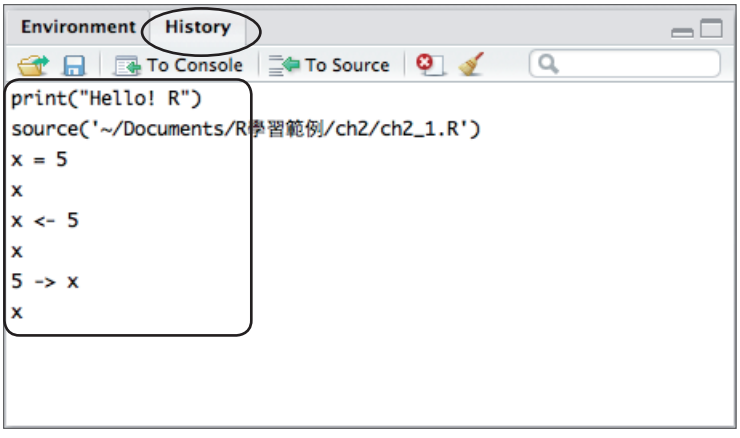
有些计算机语言，变量在使用前要先定义，R 语言则不需先定义，可在程序中直接设定使用，如本节实例所示。

2-5 Workspace 窗口

在 Workspace 窗口中，如果单击“Environment”标签，则可以看到至今所使用的对象变量及此对象变量的值，如下图所示。



如果单击“History”标签，则可以看到 Console 窗口的所有执行命令的记录，如下图所示。



此外，若在 Console 窗口输入 ls（），可以列出目前 Environment 所记录的所有对象变量，如下所示。

```
> ls()
[1] "x"
>
```

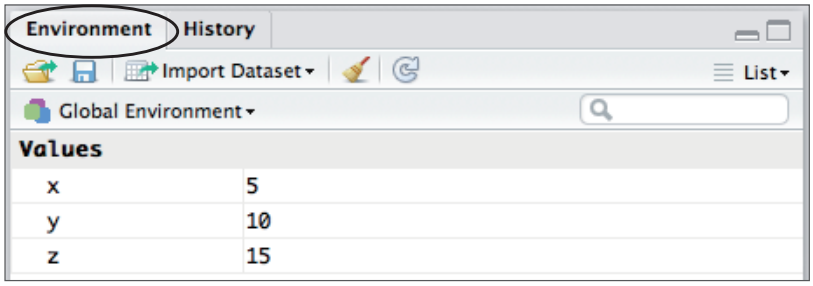
延续之前实例，增加对象变量 y，z。并设定对象变量 y 等于 10，对象变量 z 值等于对象变量 x 加上对象变量 y，如下所示。

```
> y <- 10
> z <- x + y
> z
[1] 15
>
```

此时在 Console 窗口输入 ls（），可以看到有 3 个对象变量，x、y 和 z，如下所示。

```
> ls( )
[1] "x" "y" "z"
>
```

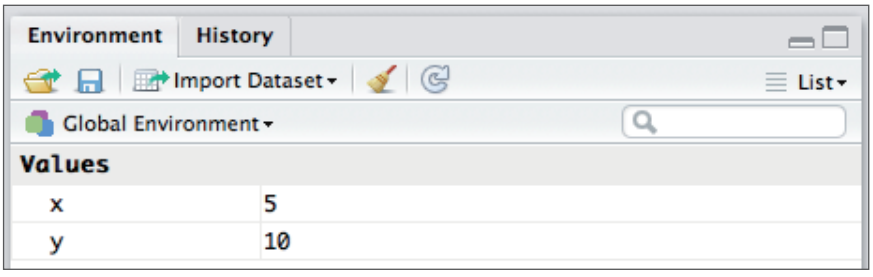
如果检查 Workspace 窗口，则可以看到这 3 个对象变量及其值，如下图所示。



使用 R 时，如果某个对象变量不再使用，则可以使用 rm () 函数，将此对象变量删除。下列是删除 z 对象变量的实例及验证结果。

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

此时 Workspace 窗口内的 z 对象变量也不再出现了，如下图所示。



2-6 结束 RStudio

在 Console 窗口，输入 q ()，来结束使用 RStudio，如下所示。

```
> q()
Save workspace image to ~/.RData? [y/n/c]:
```

- y：表示将上述对象变量和对象变量的值存储在“.RData”文件，未来只要启动 RStudio，此“.RData”文件均会被加载到 Workspace 窗口。如果你将此文件在文件夹中删除，则重新启动 RStudio 时，Workspace 窗口的内容就会是空白。2-7-2 节会介绍此文件，供未来使用。
- n：表示不储存。
- c：表示取消。

也可以执行 RStudio 窗口的 File/Quit RStudio 命令，结束使用 RStudio，效果相同。

2-7 保存工作成果

在正式谈保存工作成果前，笔者将先介绍另一个函数 getwd ()，用这个函数可以了解目前工作的文件夹，相当于未来保存工作成果的文件夹。下列是笔者计算机的执行结果。

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

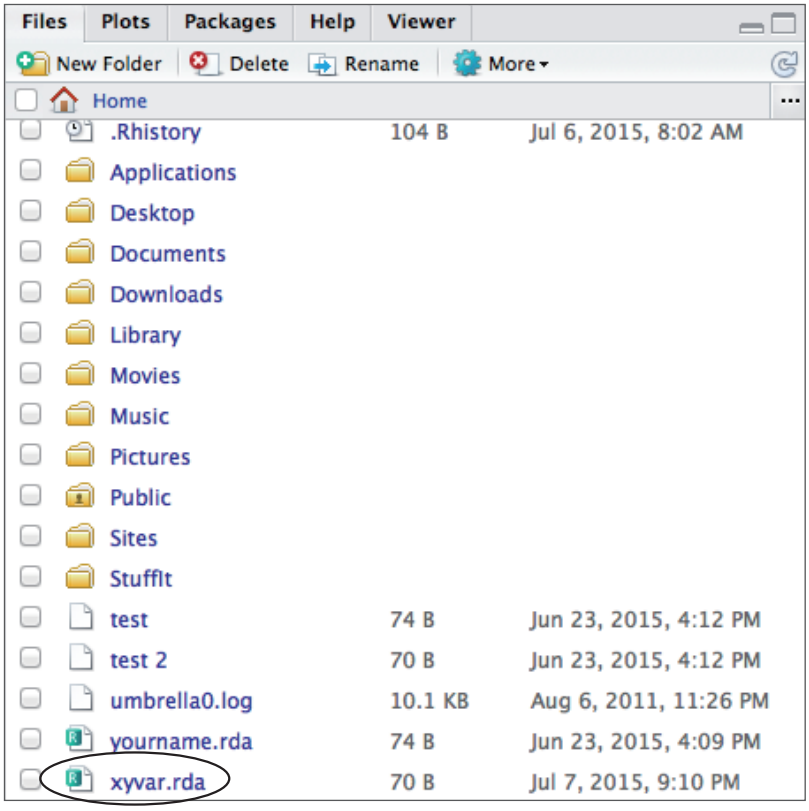
使用不同的操作系统，可能会有不同的结果。

2-7-1 使用 save () 函数保存工作成果

下列是笔者将 x 和 y 对象变量保存在“xyvar.rda”文件中的运行实例。

```
> save(x, y, file = "xyvar.rda")
>
```

执行完后，无任何确认信息，不过，可以在 RStudio 窗口右下方的 Files/Plots 窗口看到此“xyvar.rda”文件，如下图所示。



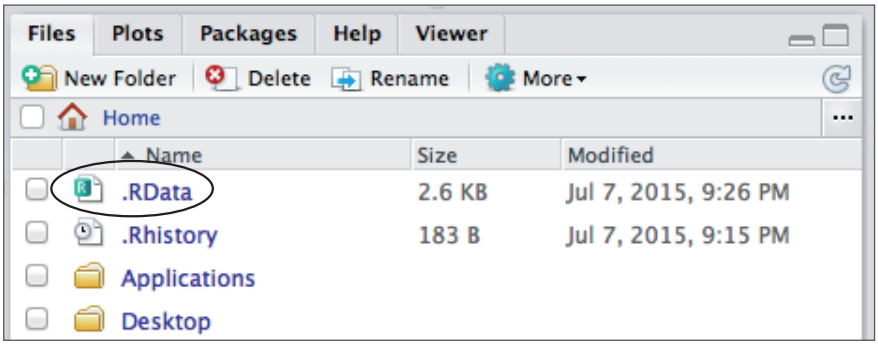
当在窗口中看到上述文件时，表示保存对象变量 x 和 y 的操作成功了。

2-7-2 使用 save.image () 函数保存 Workspace

使用 save.image () 函数可以将整个 Workspace 保存在系统默认的“.RData”文件内，如下所示。

```
> save.image()
>
```

上述命令被执行后可以得到下图所示的执行结果。



2-7-3 下载之前保存的工作

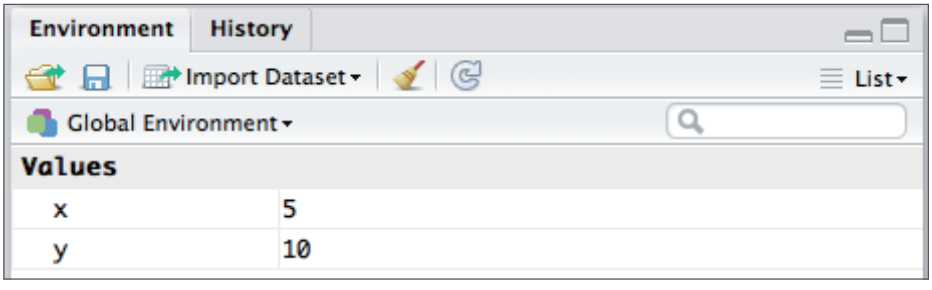
请先使用 rm () 函数清除 Workspace 窗口的对象变量值。下列命令是清除对象变量 x 和 y 的值。

```
> rm(x)
> rm(y)
>
```

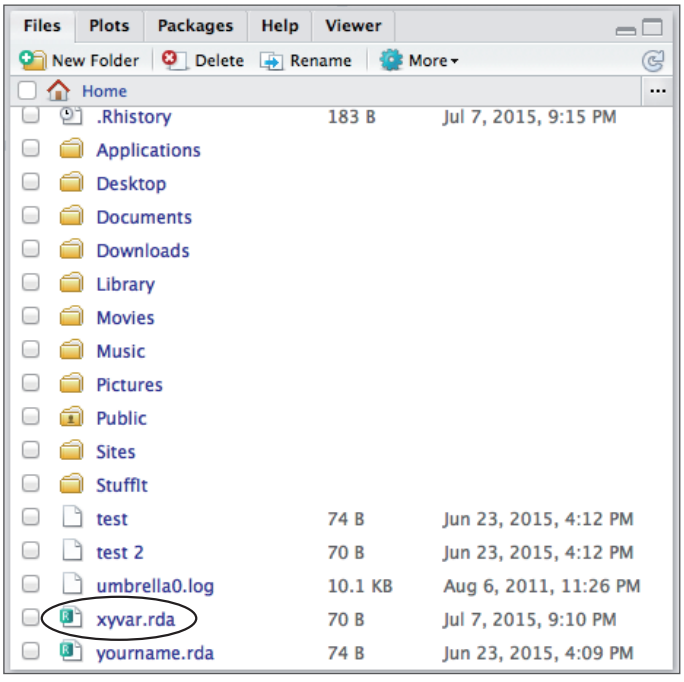
方法 1：使用 load () 函数，直接下载先前保存的值，如下所示。

```
> load("xyvar.rda")
>
```

如果此时检查 Workspace 窗口，则可以得到下列结果，窗口中列出对象变量 x 和 y 的值。



方法 2：也可以直接单击 RStudio 窗口右下方 Files/Plots 窗口的“xyvar.rda”文件，即可下载之前储存的工作，如下图所示。

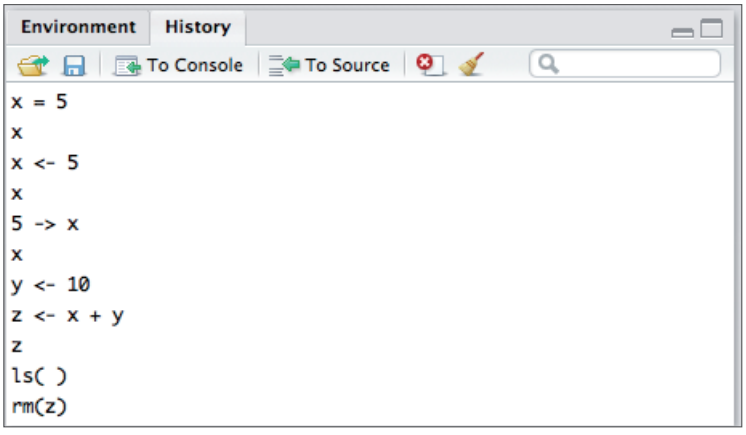


笔者在 2-7-2 节有介绍，可使用 save.image () 将工作储存在“.RData”，其实也可以使用上述方法，双击“RData”，下载所储存的工作。

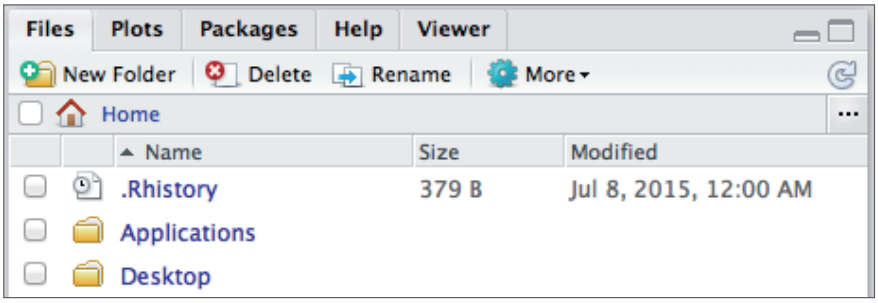
2-8 历史记录

启动 RStudio 后，基本上所有执行过的命令均会被记录在 Workspace 窗口的 History 标签选项内，如下图所示。有时为了方便，不想太麻烦重新输入命令，可以单击此区执行过的指令，然后执行下列两个操作。

- To Console：将所单击的命令，重载到 Console 窗口。
- To Source：将所单击的命令，重载到 Source Editor 窗口。



这可方便查阅所使用过的命令，或重新运行。如果你想将此历史记录保存，可以使用savehistory（）函数。然后此历史记录会被存入“.Rhistory”文件内。你可以通过查看 RStudio 窗口右下方的 Files/Pilots 窗口，看到此文件。



如果想用其他名称储存此历史档，则可使用下列方式。下列是将历史文件储存至“ch2_2.Rhistory”文件内。

```
> savehistory(file = "ch2_2.Rhistory")
>
```

如果想加载“.Rhistory”，则可以使用下列命令。

```
> loadhistory( )
>
```

如果想加载特定的历史文件，例如先前储存的“ch2_2.Rhistory”，则可以使用下列命令。

```
> loadhistory(file = "ch2_2.Rhistory")
>
```

2-9 程序注释

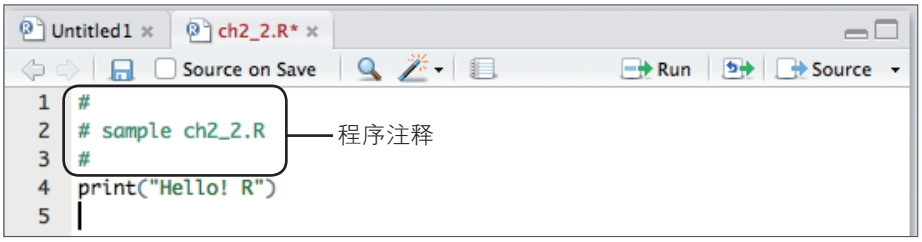
程序注释的主要功能是让你所设计的程序可读性更高，更容易了解。在企业工作，一个实用的程序可以很轻易地超过几千或上万列，此时你可能需要设计好几个月。程序加上注释，可方便你或他人，未来较便利地了解程序内容。

不论是使用直译器或是 R 程序文件中，“#”符号右边的文字，皆被称为程序注释，R 语言的直译器或编译程序皆会忽略此符号右边的文字。可参考下列实例。

```
> x <- 5
> x   # print x
[1] 5
>
```

上述第二列“#”符号右边的文字，“print x”，是此程序的注释。下图所示的是 R 程序文件的一

个实例。



上述程序实例“ch2_2.R”的前 3 行，由于有“#”符号，代表是程序注释，在此笔者特别注明，这是程序 ch2_2.R，相当于本章 2–3 节中的第二个程序实例。所以真正的程序只有第 4 行。

本章习题

一、判断题

（　）1. RStudio 的 Console 窗口主要是编辑 R 语言程序代码，储存，最后再执行的窗口。

（　）2. R 语言有支持直译器（Interpreter），可以在 Console 窗口直接输入命令，同时获得执行结果。

（　）3. 在 Workspace 窗口，如果选择 Environment 标签，则可以在此看到 Console 窗口的所有执行指令的记录。

（　）4. 一个完整的 R 程序，即使在 Source Editor 区编辑，其执行的非图形数据结果，将是在 Console 窗口中显示。

（　）5. 下列 3 个命令的执行结果是一样的。

```
> x = 10
```

或

```
> x <- 10
```

或

```
> 10 -> x
```

二、单选题

（　）1. 下列哪一个符号是程序注释符号？

A. %　　B. @　　C. #　　D. ~

（　）2. 如果我们想使用 R 语言的直译功能，可以在下列哪一个窗口输入命令？

A. Console 窗口　　B. Source Editor 窗口

C. Workspace 窗口　　D. Files/Plots 窗口

（　）3. 可以在以下哪一个窗口看到所有变量名称和它的内容？

A. Console 窗口　　B. Source Editor 窗口

C. Workspace 窗口　　D. Files/Plots 窗口

（　）4. 下列哪一个符号不是 R 语言的等号符号？

A. =　　B. <–　　C. –>　　D. #

（　）5. 下列哪一个函数可以在 Console 窗口列出所有变量数据？

A. ls ()　　B. rm ()　　C. q ()　　D. getwd ()

（　）6. 下列哪一个函数可以将整个 Workspace 保存在系统默认的“.RData”文件内？

A. save ()　　B. save.image ()　　C. load ()　　D. savehistory ()

三、多选题

(　　) 1. 哪几个函数可以保存 Console 窗口执行过的命令？（选择两项）

A. save ()

B. save.image ()

C. load ()

D. savehistory ()

E. getwd ()

四、实际操作题（如果题目有描述不周详时，请自行假设条件）

1. 请研究 RStudio 窗口右上角的 Workspace 窗口，说明下列标签的功能。

(1) Environment。

(2) History。

(3) To Console。

(4) To Source。

2. 请研究 RStudio 窗口右下角的 Files/Plots 窗口，说明下列标签的功能。

(1) Files。

(2) Export。

MEMO

03

CHAPTER

R 的基本数学运算

本章笔者将从为对象变量（也可简称对象）命名说起，接着介绍 R 的基本算术运算。

3-1 对象命名原则

在 2-9 节中，笔者介绍过，可以使用程序注释增加程序的可读性。在为对象命名时，如果使用适当名称，也可以让你所设计的程序可读性增加许多。R 的基本命名规则包括以下几点。

1） 下列名称是 R 语言的保留字，不可当作是对象名称。

break, else, FALSE, for, function, if, Inf, NA, NaN, next, repeat, return, TRUE, while

2） R 对英文字母大小写是敏感的，所以 basket 与 Basket，会被视为两个不同的对象。

3） 对象名称开头必须是英文字母或点号（“.”），当以点号（“.”）开头时，接续的第二个字母不可是数字。

4） 对象名称只能包含字母、数字、底线（“_”）和点号（“.”）。

笔者曾深深体会到，所设计的程序，时间一久后，常常会忘记各变量对象所代表的意义，所以除了为程序加上注释外，为对象取个好名字也是程序设计师很重要的课题。例如，假设想为 James 和 Jordon 打篮球的得分取对象名称。你可以按如下设计。

ball1——代表 James 的得分。

ball2——代表 Jordon 的得分。

上述方式简单，但时间久了，比较容易忘记。如果用下列方式命名。

basket.James——代表 James 的得分。

basket.Jordon——代表 Jordon 的得分。

相信即使几年后，你仍可了解此对象所代表的意义。在上述命名时，笔者在名称中间加上点号（“.”），在 R 语言中，这是 R 程序设计师常用的命名方式，又称点式风格（Dotted Style）。事实上，R 语言的许多函数皆是采用此点式命名的，例如，2-9 节所介绍的 save.image（）函数。

另外，为对象命名时也会采用驼峰式（Camel Case），将组成对象名称的每一个英文字母开头用大写。例如，my.First.Ball.Game，这样可以直接明白此对象名称的意义。

3-2 基本数学运算

3-2-1 四则运算

R 的四则运算是指加（ + ）、减（ - ）、乘（ * ）和除（ / ）。

实例 ch3_1：加法与减法运算实例。

```
> x1 = 5 + 6        #将5加6设定给对象x1
> x1
[1] 11
> x2 = x1 + 10      #将x1加10设定给对象x2
> x2
[1] 21
> x3 = x2 - x1      #将x2减x1设定给对象x3
> x3
[1] 10
>
```

在以上赋值（也可想成等号）中，笔者故意用“=”符号，本章赋值有时候也会用“<-”，主要是用实例让读者了解 R 是支持这两种赋值符号的。从第四章起笔者将统一使用“<-”当作赋值符号。

实例 ch3_2：乘法与除法运算实例。

```
> x1 = 5
> x2 = 9
> x3 = x1 * x2      #x3等于x1乘以x2
> x3
[1] 45
> x4 = x2 / x1      #x4等于x2除以x1
> x4
[1] 1.8
>
```

3-2-2 余数和整除

余数（mod）所使用的符号是“%%”，可计算出除法运算中的余数。整除所使用的符号是“%/%”，是指在除法运算中只保留整数部分。

实例 ch3_3：余数和整除运算实例。

```
> x = 9 %% 5        #计算9除以5的余数
> x
[1] 4
> x = 9 %/% 2       #计算9除以2所得的整数部分
> x
[1] 4
>
```

3-2-3 次方或平方根

次方的符号是“**”或“^”，平方根的计算是使用函数 sqrt ()。

实例 ch3_4：平方、次方和平方根运算实例。

```
> x = 3 ** 2        #计算3的平方
> x
[1] 9
> x = 3 ^ 2         #计算3的平方
> x
[1] 9
> x = 8 ^ 3         #计算8的3次方
> x
[1] 512
> x = sqrt(64)      #计算64的平方根
> x
[1] 8
> x = sqrt(8)       #计算8的平方根
> x
[1] 2.828427
>
```

3–2–4　绝对值

绝对值的函数名称是 abs（），不论函数内的值是正数或负数，结果均是正数。

实例 ch3_5：绝对值运算实例。

```
> abs(10)           #计算10的绝对值
[1] 10
> x = 5.5
> y = abs(x)        #计算x的绝对值
> y
[1] 5.5
> x = -7
> y = abs(x)        #计算x的绝对值
> y
[1] 7
>
```

3–2–5　exp（）与对数

exp（）是指自然数 e 的 x 次方，其中 e 的近似值是 2.718282。

实例 ch3_6：exp（）运算实例。

```
> x = exp(1)        #可列出自然数e的值
> x
[1] 2.718282
> x = exp(2)        #可列出自然数e的2次方
> x
[1] 7.389056
> x = exp(0.5)      #可列出自然数e的0.5次方
> x
[1] 1.648721
>
```

对数有以下两种类型。

1） 以自然数 e 为底的对数，$\log_e x = \ln x$，语法是 log ()。

2） 一般基底的对数，$\log_m x$ ，语法是 log (x, m)。如果基底是 10，也可使用另一个对数函数 log10 () 取代。

实例 ch3_7：不同基底的对数运算实例。

```
> x = log(2)        #计算以自然数e为底的对数值
> x
[1] 0.6931472
> x = log(2, 10)    #计算以自然数10为底的对数值
> x
[1] 0.30103
> x = log10(2)      #计算以自然数10为底的对数值
> x
[1] 0.30103
> x = log(2, 2)     #计算以自然数2为底的对数值
>
```

exp () 和 log () 也可称互为反函数。

3–2–6 科学符号 e

科学符号是用 e 表示，例如数字 12 800，实际等于“1.28 * 10^4”，也可以用“1.28e4”表示。

实例 ch3_8：科学符号的运算实例。

```
> x <- 1.28 * 10^4
> x
[1] 12800
> x <- 1.28e4
> x
[1] 12800
>
```

数字 0.00365，实际等于“3.65 * 10^–3”，也可以用“3.65e–3”表示。

实例 ch3_9：另一个科学符号的运算实例。

```
> x <- 3.65 * 10^-3
> x
[1] 0.00365
> x <- 3.65e-3
> x
[1] 0.00365
>
```

当然也可以直接使用科学符号执行四则运算。

实例 ch3_10：直接使用科学符号的运算实例。

```
> x <- 6e5 / 3e2
> x
[1] 2000
>
```

上述的代码表示 600000 除以 300。

3–2–7　圆周率与三角函数

圆周率就是指 pi。pi 是系统默认的参数，其近似值是 3.141593。

实例 ch3_11：列出 pi 值的实例。

```
> pi
[1] 3.141593
>
```

R 语言所提供的三角函数有许多，例如，sin ()、cos ()、tan ()、asin ()、acos ()、atan ()、sinh ()、cosh ()、tanh ()、asinh ()、acos ()、atan ()。

实例 ch3_12：三角函数运算实例。

```
> x = sin(1.0)
> x
[1] 0.841471
> x = sin(pi / 2)
> x
[1] 1
> x = cos(1.0)
> x
[1] 0.5403023
> x = cos(pi)
> x
[1] -1
>
```

3–2–8　四舍五入函数

R 语言的四舍五入函数是 round ()。

round (x, digits = k)，表示将实数 x，以四舍五入方式，计算至第 k 位小数。另外，round () 函数中的第 2 个参数“digits =”也可以省略，直接在第 2 个参数位置输入数字。

实例 ch3_13：round () 函数的各种运用实例。

```
> x <- round(98.562, digits = 2)
> x
[1] 98.56
> x <- round(98.562, digits = 1)
> x
[1] 98.6
> x <- round(98.562, 2)
> x
[1] 98.56
> x <- round(98.562, 1)
> x
[1] 98.6
>
```

使用 round（）函数时，如果第 2 个参数是负值，表示计数是以四舍五入取整数。例如，若参数是“–2”，表示取整数至百位数。若参数是“–3”，表示取整数至千位数。

实例 ch3_14：使用 round（）函数，但 digits 参数是负值的运用实例。

```
> x <- round(1234, digits = -2)
> x
[1] 1200
> x <- round(1778, digits = -3)
> x
[1] 2000
> x <- round(1234, -2)
> x
[1] 1200
> x <- round(1778, -3)
> x
[1] 2000
>
```

signif（x, digits = k），也是一个四舍五入的函数，其中 x 是要做处理的实数，k 是有效数字的个数。例如，signif（79843.597, digits = 6），代表取 6 个数字，从左边算第 7 个数字以四舍五入的方式处理。

实例 ch3_15：signif（）函数的应用实例。

```
> x <- signif(79843.597, digits = 6)
> x
[1] 79843.6
> x <- signif(79843.597, 6)
> x
[1] 79843.6
> x <- signif(79843.597, digits =  3)
> x
[1] 79800
> x <- signif(79843.597, 3)
> x
[1] 79800
>
```

3–2–9 近似函数

R 语言有 3 个近似函数。

1） floor（x）：可得到小于等于 x 的最近整数。所以，floor（234.56）等于 234。floor（–234.45）等于 –235。

2） ceiling（x）：可得到大于等于 x 的最近整数。所以，ceiling（234.56）等于 235。ceiling（–234.45）等于 –234。

3） trunc（x）：可直接取整数。所以，trunc（234.56）等于 234。trunc（–234.45）等于 –234。

实例 ch3_16：floor（）、ceiling（）和 trunc（）函数的运用实例。

```
> x <- floor(234.56)
> x
[1] 234
> x <- floor(-234.45)
> x
[1] -235
> x <- ceiling(234.56)
> x
[1] 235
> x <- ceiling(-234.45)
> x
[1] -234
> x <- trunc(234.56)
> x
[1] 234
> x <- trunc(-234.45)
> x
[1] -234
>
```

3-2-10 阶乘

factorial（x）可以返回 x 的阶乘。

实例 ch3_17：factorial（）函数的运用。

```
> x <- factorial(3)
> x
[1] 6
> x <- factorial(5)
> x
[1] 120
> x <- factorial(7)
> x
[1] 5040
>
```

3-3 R 语言控制运算的优先级

当 R 语言碰上多种计算同时出现在一个指令内时，除了括号“()”最优先外，其余计算优先次序如下。

1） 指数。

2） 乘法、除法、求余数（%%）、求整数（%/%），依照出现顺序运算。

3） 加法、减法，依照出现顺序运算。

实例 ch3_18：R 语言控制运算的优先级的应用实例。

```
> x <- ( 5 + 6 ) * 8 - 2
> x
[1] 86
> x <- 5 + 6 * 8 - 2
> x
[1] 51
>
```

3-4 无限大 Infinity

R 语言可以处理无限大的值，使用 Inf 表示，如果是负无限大则是 –Inf。其实只要将某一个数字除以 0，就可获得无限大。

实例 ch3_19：无限大 Inf 的取得实例。

```
> x <- 5 / 0
> x
[1] Inf
>
```

将某一个数字减去无限大 Inf，可以获得负无限大 –Inf。

实例 ch3_20：负无限大 –Inf 的取得实例。

```
> x <- 10 - Inf
> x
[1] -Inf
>
```

另一个思考，如果将某一个数字除以无限大 Inf 或负无限大 –Inf，结果是多少？答案是 0。

实例 ch3_21：把 Inf 和 –Inf 当作分母的应用实例。

```
> x <- 999 / Inf
> x
[1] 0
> x <- 999 / -Inf
> x
[1] 0
>
```

判断某一个数字是否为无限大（正值无限大或负值无限大），可以使用 is.infinite（x），如果 x 是则返回逻辑值（Logical Value）TRUE，否则返回 FALSE。

实例 ch3_22：使用 is.infinite（）判断 Inf 和 –Inf 是否为正或负无限大，返回 TRUE 的实例。

```
> x <- 10 / 0
> x
[1] Inf
```

```
> is.infinite(x)
[1] TRUE
> x <- 10 - x
> x
[1] -Inf
> is.infinite(x)
[1] TRUE
>
```

实例 ch3_23：使用 is.infinite（）判断 Inf 和 –Inf 是否为正或负无限大，返回 FALSE 的实例。

```
> x <- 999
> is.infinite(x)
[1] FALSE
> x <- -99999
> is.infinite(x)
[1] FALSE
>
```

另一个有关的函数式 is.finite（x），如果数字 x 是有限的（正有限大或负有限大）则返回 TRUE，否则返回 FALSE。

实例 ch3_24：使用 is.finite（）判断一个数是否为有限大的实例。

```
> x <- 999
> is.finite(x)
[1] TRUE
> x <- -99999
> is.finite(x)
[1] TRUE
> x <- 10 / 0
> is.finite(x)
[1] FALSE
> x <- 10 - ( 10 / 0 )
> x
[1] -Inf
> is.finite(x)
[1] FALSE
>
```

在其他程序语言中，TRUE 和 FALSE 被称为布尔值（Boolean Value），但在 R 语言中，R 的开发人员将此称为逻辑值（Logical Value）。

3-5 Not a Number（NaN）

在 R 语言中，Not a Number（NaN）可以解释为非数字或无定义数字，由上一小节可知，任一数字除以 0 可得无限大，任一数字除以无限大可得 0，那无限大除以无限大呢？此时可以获得

NaN（Not a Number）。

实例 ch3_25：NaN 值的获得实例。

```
> x <- Inf / Inf
> x
[1] NaN
>
```

R 语言将 NaN 当作一个数字，可以使用 NaN 参加四则运算，但所得结果均是 NaN。

实例 ch3_26：NaN 值的四则运算实例。

```
> x <- NaN + 999
> x
[1] NaN
> x <- NaN * 2
> x
[1] NaN
>
```

使用 is.nan（x）函数，可检测 x 是否为 NaN，如果是则返回 TRUE，否则返回 FALSE。

实例 ch3_27：当 is.nan（）函数的参数是 NaN 时的运算实例。

```
> x <- Inf / Inf
> x
[1] NaN
> is.nan(x)
[1] TRUE
> y <- 999
> is.nan(y)
[1] FALSE
>
```

另外，对于 NaN 而言，无论使用 is.finite（）还是 is.infinite（）判断，均传回 FALSE。

实例 ch3_28：为 is.finite（）和 is.infinite（）函数的参数是 NaN 时的运算实例。

```
> x <- Inf / Inf
> x
[1] NaN
> is.finite(x)
[1] FALSE
> is.infinite(x)
[1] FALSE
>
```

3-6 Not Available（NA）

Not Available 也可被称为缺失值 NA，我们可以将 NA 当作一个有效数值，甚至可以将此值应用在四则运算中，不过，通常计算结果是 NA。

实例 ch3_29：缺失值 NA 的运算实例。

```
> x <- NA
> y <- NA + 100
> y
[1] NA
> z <- NA / 10
> z
[1] NA
>
```

R 语言提供的 is.na（x）函数可判断 x 是否为 NA，如果是则返回 TRUE，否则返回 FALSE。

实例 ch3_30：is.na（）函数的参数是缺失值 NA 和一般值的运算实例。

```
> x <- NA
> is.na(x)
[1] TRUE
> x <- 1000
> is.na(x)
[1] FALSE
>
```

对于 NaN 而言，使用 is.na（）判断，可以得到 TRUE。

实例 ch3_31：is.na（）函数的参数是 NaN 的运算实例。

```
> x <- Inf / Inf
> x
[1] NaN
> is.na(x)
[1] TRUE
>
```

本章习题

一、判断题

（　　）1. 有以下两个命令。

```
> x1 <- 9 %% 5
> x2 <- 9 %/% 2
```

上述两个命令被执行后，x1 和 x2 的值是相同的，均是 4。

（　　）2. 有以下两个命令。

```
> x1 <- 2 ^ 3
> x2 <- sqrt(64)
```

上述两个命令被执行后，x1 和 x2 的值是相同的，均是 8。

（　　）3. 有以下两个命令。

```
> x1 <- round(88.882, digits = 2)
> x2 <- round(88.882, 2)
```

上述两个命令被执行后，x1 和 x2 的值是相同的，均是 88.88。

（　　）4. 有如下命令。

```
> x <- round(1560.998, digits = -2)
```

上述命令被执行后，x 的值是 1600。

（　　）5. 有如下命令。

```
> x <- factorial(3)
```

上述命令被执行后，x 的值是 8。

（　　）6. 有如下命令。

```
> x <- 10 / Inf
```

上述命令被执行后，x 的值是 0。

（　　）7. 有以下两个命令。

```
> x <- 999 / 0
> is.infinite(x)
```

上述命令的执行结果是 FALSE。

（　　）8. 有如下命令。

```
> x <- Inf / Inf
```

上述命令被执行后，x 的值是 1。

（　　）9. 有以下两个命令。

```
> x <- NA + 999
> is.na(x)
```

上述命令的执行结果是 TRUE。

（　　）10. 有以下两个命令。

```
> x <- 888 * 999
> is.finite(x)
```

上述命令的执行结果是 TRUE。

二、单选题

（　　）1. 下列哪一个是 R 语言不合法的变量名称？

A. x3　　B. x.3　　C. .x3　　D. 3.x

（　　）2. 以下命令会得到哪种数值结果？

```
> -3 + 2 ** 3 - 4^2 / 8
```

A. [1] 4　　B. [1] 2　　C. [1] 3　　D. [1] 1

（　　）3. 以下命令会得到哪种数值结果？

```
> round(pi, 2)
```

A. [1] 3.1415926　　B. [1] pi　　C. [1] 3.14　　D. [1] 3

（　　）4. 以下命令会得到哪种数值结果？

```
> 36 ** 0.5
```

A. [1] 18　　B. [1] 6　　C. [1] 9　　D. [1] 3

（　　）5. 以下命令会得到哪种数值结果？

```
> signif(5678.778, 6)
```

A. [1] 5678.78　　B. [1] 5678.77

C. [1] 5678.778　　D. [1] 5678.778000

（　　）6. 以下命令会得到哪种数值结果？

```
> floor(789.789)
```

A. [1] 789.8　　B. [1] 789.789　　C. [1] 789　　D. [1] 790

（　　）7. 以下命令会得到哪种数值结果？

```
> x <- Inf / 1000
```

A. [1] 0　　B. [1] Inf　　C. [1] NA　　D. [1] NaN

三、多选题

（　　）1. 下列哪些命令的执行结果是 TRUE？（选择两项）

A.
```
> x <- Inf - Inf
> is.infinite(x)
```

B.
```
> x <- Inf + Inf
> is.infinite(x)
```

C.
```
> x <- Inf + 1010
> is.na(x)
```

D.
```
> x <- Inf / Inf
> is.nan(x)
```

E.
```
> x <- 1010
> is.nan(x)
```

四、实际操作题（如果题目有描述不周详的，请自行假设条件）

1. 求 99 的平方、立方和平方根。
2. x = 345.678，将 x 放入 round ()、signf ()，使用默认值测试，并列出结果。
3. 重复上一习题，将参数 digits 依次从 –2 设置到 2，并列出结果。
4. x = 674.378，将 x 放入 floor ()、ceil () 和 trunc ()，使用默认值测试，并列出结果。
5. 重复上一习题，将 x 改为负值 –674.378，并列出结果。
6. 计算下列命令的结果。

   (1) Inf + 100。

   (2) Inf – Inf + 10。

   (3) NaN + Inf。

   (4) Inf– NaN。

   (5) NA + Inf。

   (6) Inf – NA。

   (7) NaN + NA。
7. 将上述数据（a–g）的执行结果用下列函数测试并列出结果。

   (1) is.na ()。

   (2) is.nan ()。

   (3) is.finite ()。

   (4) is.infinite ()。

MEMO

04

CHAPTER

向量對象運算

R 语言最重要的特色是向量（Vector）对象的概念。如果你学过其他计算机语言，应该知道一维数组（Array）的概念，其实所谓的向量对象就是类似一组一维数组的数据，在此组数据中，每个元素的数据类型是一样的。不过向量对象的使用比其他高级语言灵活太多了，R 的开发团队将此一维数组数据称为向量（Vector）对象。

说穿了，R 语言就是一种处理向量的语言。

其实 R 语言中最小的工作单位是向量对象，至于前面章节笔者当作范例使用的一些对象变量，从技术上讲可将那些对象变量看作是一个只含一个元素的向量对象变量。至今为止，在输出每一个数据时，首先出现的是“[1]”，中括号内的“1”表示接下来是从对象的第 1 个元素开始输出的。对数学应用而言，向量对象元素大都是数值数据型的，R 的更重要的功能是向量对象元素可以是其他数据型的，本书将在以后章节中一一介绍。

4-1 数值型的向量对象

数值型的向量对象可分为规则型的数值向量对象或不规则型的数值向量对象。

4-1-1 建立规则型的数值向量对象应使用序列符号

从起始值到最终值，每次递增 1，如果是负值则每次增加 –1。例如从 1 到 5，可用 1:5 的方式表达。从 11 到 16，可用 11:16 的方式表达。在“1:5”或“11:16”的表达式中的“:”符号，即冒号，在 R 语言中称其为序列符号（Sequence）。

实例 ch4_1：使用序列号“:”建立向量对象。

```
> x <- 1:5          #设定向量变量对象包含1到5共5个元素
> x
[1] 1 2 3 4 5
> x <- 11:16        #设定向量变量对象包含11到16共6个元素
> x
[1] 11 12 13 14 15 16
>
```

这种方式也可以应用于负值，每次增加 –1。例如，从 –3 到 –7，可用 –3:–7 的方式表达。

实例 ch4_2：使用序列号建立含负数的向量对象。

```
> x <- -3:-7         #设定向量变量对象包含-3到-7共5个元素
> x
[1] -3 -4 -5 -6 -7
>
```

同理，这种方式也可以应用于实数，每次增加正 1 或 –1。

实例 ch4_3：使用序列号建立实数的向量对象。

```
> x <- 1.5:5.5        #设定向量变量对象包含1.5到5.5共5个元素
> x
[1] 1.5 2.5 3.5 4.5 5.5
> x <- -1.8:-3.8      #设定向量变量对象包含-1.8到-3.8共3个元素
> x
[1] -1.8 -2.8 -3.8
>
```

在建立向量对象时，如果写成 1.5:4.7，结果会如何呢？这相当于建立含下列元素的向量对象，1.5、2.5、3.5、4.5 共 4 个元素，至于多余部分即 4.5 至 4.7 之间的部分则可不理会。若向量对象的元素为负值时，依此类推。

实例 ch4_4：另一个使用序列号建立实数的向量对象。

```
> x <- 1.5:4.7        #设定向量变量对象包含1.5到4.5共4个元素
> x
[1] 1.5 2.5 3.5 4.5
> x <- -1.3:-5.2      #设定向量变量对象包含-1.3到-4.3共4个元素
> x
[1] -1.3 -2.3 -3.3 -4.3
>
```

4–1–2 简单向量对象的运算

向量对象的一个重要功能是向量对象在执行运算时，向量对象内的所有元素将同时执行运算。

实例 ch4_5：将每一个元素加 3 的执行情形。

```
> x <- 1:5
> y <- x + 3
> y
[1] 4 5 6 7 8
>
```

一个向量对象也可以与另一个向量对象相加。

实例 ch4_6：向量对象相加的实例。

```
> x <- 1:5
> y <- x + 6:10       #设定x向量加6:10向量，结果设定给向量y
> y
[1]  7  9 11 13 15
>
```

读至此节，相信各位读者一定已经感觉到 R 语言的强大功能了，如果上述指令使用非向量语言，则需使用循环指令处理每个元素，要好几个步骤才可完成。在执行向量对象元素的运算时，也可以处理不相同长度的向量对象运算，但先决条件是较长的向量对象的长度是较短的向量对象的长度的倍数。如果不是倍数，则会出现错误信息。

实例 ch4_7：不同长度的向量对象相加，出现错误信息的实例。

```
> x <- 1:5
> y <- x + 5:8
Warning message:
In x + 5:8 : 较长的对象长度并非较短对象长度的倍数
```

由于上述较长的向量对象有 5 个元素，较短向量对象有 4 个元素，所以较长向量的长度不是较短向量的长度的倍数，因此最后执行后出现警告信息。

实例 ch4_8：不同长度的向量对象相加，较长向量对象的长度是较短向量对象的长度的倍数的运算实例。

```
> x <- 1:3
> y <- x + 1:6
> y
[1] 2 4 6 5 7 9
>
```

上述的运算规则是，向量对象 y 的长度与较长的向量对象的长度相同，其长度是 6，较长向量对象的第 1 个元素与 1:3 的 1 相加，较长向量对象的第 2 个元素与 1:3 的 2 相加，较长向量对象的第 3 个元素与 1:3 的 3 相加，较长向量对象的第 4 个元素与 1:3 的 1 相加，较长向量对象的第 5 个元素与 1:3 的 2 相加，较长向量对象的第 6 个元素与 1:3 的 3 相加。未来如果碰上不同倍数的情况，运算规则可依此类推。

实例 ch4_9：下列是另一个不同长度向量对象相加的实例。

```
> x <- 1:5
> y <- 5
> x + y
[1]  6  7  8  9 10
>
```

在上述实例中，x 向量对象有 5 个元素，y 向量对象有 1 个元素，碰上这种加法，相当于每个 x 向量元素均加上 y 向量的元素值。过去的实例，在输出时，笔者均直接输入向量对象变量，即可在 Console 窗口打印此向量对象变量的值，在此例中，可以看到第 3 列，即使仍是一个数学运算，Console 窗口仍将打印此数学运算的结果。

4–1–3　建立向量对象函数 seq（）

seq（）函数可用于建立一个规则型的数值向量对象，它的使用格式如下所示。

seq（ from, to, by = width, length.out = numbers ）

上述 from 是数值向量对象的起始值，to 是数值向量对象的最终值，by 则指出每个元素的增值。如果省略 by 参数，同时没有 length.out 参数存在，则增值是 1 或 –1。length.out 参数字段可设定 seq（）函数所建立的元素个数。

实例 ch4_10：使用 seq（）建立规则型的数值向量对象。

```
> seq(1, 9)                    #建立1:9向量
[1] 1 2 3 4 5 6 7 8 9
> seq(1, 9, by = 2)            #建立1至9间增值为2的向量
[1] 1 3 5 7 9
> seq(1, 9, by = pi)           #建立1至9间增值为pi的向量
[1] 1.000000 4.141593 7.283185
> seq(1.5, 4.5, by = 0.5)      #建立1.5至4.5间增值为0.5的向量
[1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5
> seq(1, 9, length.out = 5)    #建立1至9间元素个数为5的向量
[1] 1 3 5 7 9
>
```

4–1–4 连接向量对象函数 c（）

c（）函数中的 c 是 concatenate 的缩写。这个函数并不是一个建立向量对象的函数，只是一个将向量元素连接起来的函数。

实例 ch4_11：使用 c（）函数建立一个简单的向量对象。

```
> x <- c(1, 3, 7, 2, 9)        #一个含5个元素的向量
> x
[1] 1 3 7 2 9
>
```

上述 x 是一个向量对象，共有 5 个元素，分别是 1、3、7、2、9。

适当地为变量取一个容易记的变量名称，可以增加程序的可读性。例如，我们想建立 NBA 球星 Lin，2016 年前 6 场赛季进球数的向量对象，那么假设他的每场进球数如下所示。

7, 8, 6, 11, 9, 12

此时可用 baskets.NBA2016.Lin 当变量名称，相信这样处理后，即使程序放再久，也可以轻易了解程序内容。

实例 ch4_12：建立 NBA 球星进球数的向量对象。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> baskets.NBA2016.Lin
[1]  7  8  6 11  9 12
>
```

如果球星 Lin 的进球皆是 2 分球，则他每场得分如下。

实例 ch4_13：计算 NBA 球星的得分。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> scores.NBA2016.Lin <- baskets.NBA2016.Lin * 2
> scores.NBA2016.Lin
[1] 14 16 12 22 18 24
>
```

假设队友 Jordon 前 6 场进球数分别是 10, 5, 9, 12, 7, 11，我们可以用如下方式计算每场两个人

的得分总计。

实例 ch4_14：计算 NBA 球星 Lin 和 Jordon 的每场总得分。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> baskets.NBA2016.Jordon <- c(10, 5, 9, 12, 7, 11)
> total <- ( baskets.NBA2016.Jordon + baskets.NBA2016.Lin ) * 2
> total
[1] 34 26 30 46 32 46
>
```

先前介绍可以使用 c（）函数，将元素连接起来，其实也可以将两个向量对象连接起来，下面是将 Lin 和 Jordon 进球连接起来，结果是一个含 12 个元素的向量对象的实例。

实例 ch4_15：使用 c（）函数建立向量对象，其中 c（）函数内有多个向量对象参数。

```
> all.baskets.NBA2016 <- c(baskets.NBA2016.Lin, baskets.NBA2016.Jordon)
> all.baskets.NBA2016
 [1]  7  8  6 11  9 12 10  5  9 12  7 11
>
```

从上述执行结果可以看到，c（）函数保持了每个元素在向量对象内的顺序，这个功能很重要，因为未来我们要讲解如何从向量对象中存取元素值。

4-1-5　重复向量对象函数 rep（）

如果向量对象内某些元素是重复的，则可以使用 rep（）函数建立这类型的向量对象，它的使用格式如下所示。

rep（x, times = 重复次数 , each = 每次每个元素的重复次数 , length.out = 向量长度）

如果 rep（）函数内只含有 x 和 times 参数，则“times =”参数可省略。

实例 ch4_16：使用 rep（）函数建立向量对象的应用。

```
> rep(5, 5)                    #重复向量元素5，共5次
[1] 5 5 5 5 5
> rep(5, times = 5)            #重复向量元素5，共5次
[1] 5 5 5 5 5
> rep(1:5, 3)                  #重复向量1:5，共3次
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:3, times = 3, each = 2)    #重复向量1:3，共3次，每次 每个元素出现2次
 [1] 1 1 2 2 3 3 1 1 2 2 3 3 1 1 2 2 3 3
> rep(1:3, each = 2, length.out = 8)   #重复向量1:3，每个元素出现2次，每次向量元素个数是8
[1] 1 1 2 2 3 3 1 1
>
```

4-1-6　numeric（）函数

numeric（）也是一个建立函数，主要是可用于建立一个固定长度的向量对象，同时向量对象元素的默认值是 0。

实例 ch4_17：建立一个含 10 个元素的向量对象，同时这些向量对象元素的值皆为 0。

```
> x <- numeric(10)                #建立一个含10个元素值为0的向量
> x                               #验证结果
 [1] 0 0 0 0 0 0 0 0 0 0
>
```

4–1–7 程序语句跨行的处理

在本章 4–1–5 节的最后一个实例中，可以很明显看到 rep（）函数包含说明文字已超出一行，其实 R 语言是可以识别这行的命令未完，下一列是相同行的。除了上述情况外，下列是几种可能发生程序跨行的情况。

1） 该行以数学符号（ +、–、*、/ ）作结尾，此时 R 语言的编译程序会知道下一行是接续此行的。

实例 ch4_18：以数学符号作结尾，了解程序语句跨行的处理。

```
> all.baskets.NBA2016 <- baskets.NBA2016.Jordon +
+                        baskets.NBA2016.Lin
> all.baskets.NBA2016
[1] 17 13 15 23 16 23
>
```

2） 使用左括号“(”，R 编辑器会知道在下一行出现的片断数据是同一括号内的命令，直至出现右括号“)”，才代表命令结束。

实例 ch4_19：使用左括号“(”和右括号“)”，了解程序跨列的处理。

```
> x <- rep(1:5, times = 2,)
> x <- rep(1:5, times = 2,
+          each = 2)
> x
 [1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5
>
```

3） 字符串是指双引号间的文字字符，在设定字符串时，如果有了第一个双引号，但尚未出现第二个双引号，R 语言编辑器可以知道下一行出现的字符串是同一字符串向量变量的数据，但此时换行字符“/n”将被视为字符串的一部分。

有关字符串数据的概念，将在 4–4 节说明。

实例 ch4_20：使用字符串，了解程序语句跨行的处理。

```
> coffee.Knowledge <- "Coffee is mainly produced
+ in frigid regions."
> coffee.Knowledge
[1] "Coffee is mainly produced\nin frigid regions."
>
```

4-2 常见向量对象的数学运算函数

研读至此，如果你学过其他高级计算机语言，你会发现向量对象变量已经取代了一般计算机程序语言的变量，这是一种新的思维，同时如果你阅读本节的常用向量对象的数学运算函数后，你将发现为何 R 这么受到欢迎。

1. 常见运算函数

sum（）：可计算所有元素的和。

max（）：可计算所有元素的最大值。

min（）：可计算所有元素的最小值。

mean（）：可计算所有元素的平均值。

实例 ch4_21：sum（）、max（）、min（）和 mean（）函数的应用。

```
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> sum(baskets.NBA2016.Lin)       #计算Lin的总进球数
[1] 53
> max(baskets.NBA2016.Lin)       #计算Lin的最高进球数
[1] 12
> min(baskets.NBA2016.Lin)       #计算Lin的最低进球数
[1] 6
> mean(baskets.NBA2016.Lin)      #计算Lin的平均进球数
[1] 8.833333
>
```

此外，这几个函数也可以在括号内放上几个向量对象变量执行运算。

实例 ch4_22：sum（）、max（）和 min（）函数的参数含有多个向量对象变量的应用。

```
> baskets.NBA2016.Jordon <- c(10, 5, 9, 15, 7, 11)
> baskets.NBA2016.Lin <- c(7, 8, 6, 11, 9, 12)
> sum(baskets.NBA2016.Lin, baskets.NBA2016.Jordon)  #计算2人的总进球数
[1] 110
> max(baskets.NBA2016.Lin, baskets.NBA2016.Jordon)  #计算2人的最高进球数
[1] 15
> min(baskets.NBA2016.Lin, baskets.NBA2016.Jordon)  #计算2人的最低进球数
[1] 5
>
```

2. prod（）函数

prod（）：计算所有元素的积。

实例 ch4_23：使用 prod（）执行阶乘的运算。

```
> prod(1:5)          #计算从1乘到5，相当于factorial(5)
[1] 120
>
```