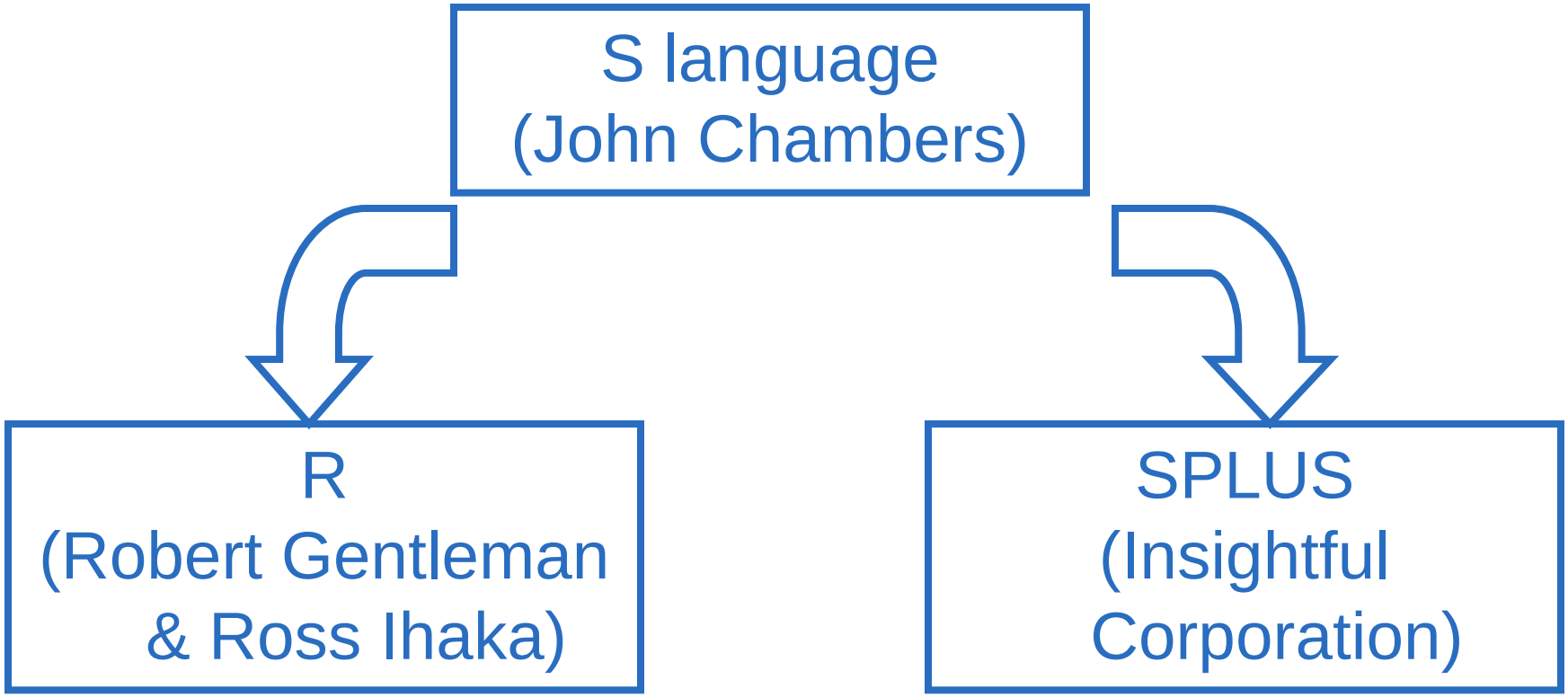
Chapter 1
Introduction to R

R language

- R is a programming language and software environment for statistical computing and graphics
- R becomes a de facto standard among statisticians for the development of statistical software
- R is widely used for statistical software development and data analysis

History of R


S language
(John Chambers)
R
(Robert Gentleman
& Ross Ihaka)
SPLUS
(Insightful
Corporation)

Installation of R

- R is free; its source code is freely available
- To download R, go to www.r-project.org, follow the direction on my handout

R System

- The R System consists of two conceptual parts:
  - The base R system – That is what you want to install R for the first time
  - Contributed packages

- The base R system contains the base package which is required to run R and contains the most fundamental functions
- The main installation will install R and a popular set of add-on libraries
- You can install libraries from CRAN or the Bioconductor

R System

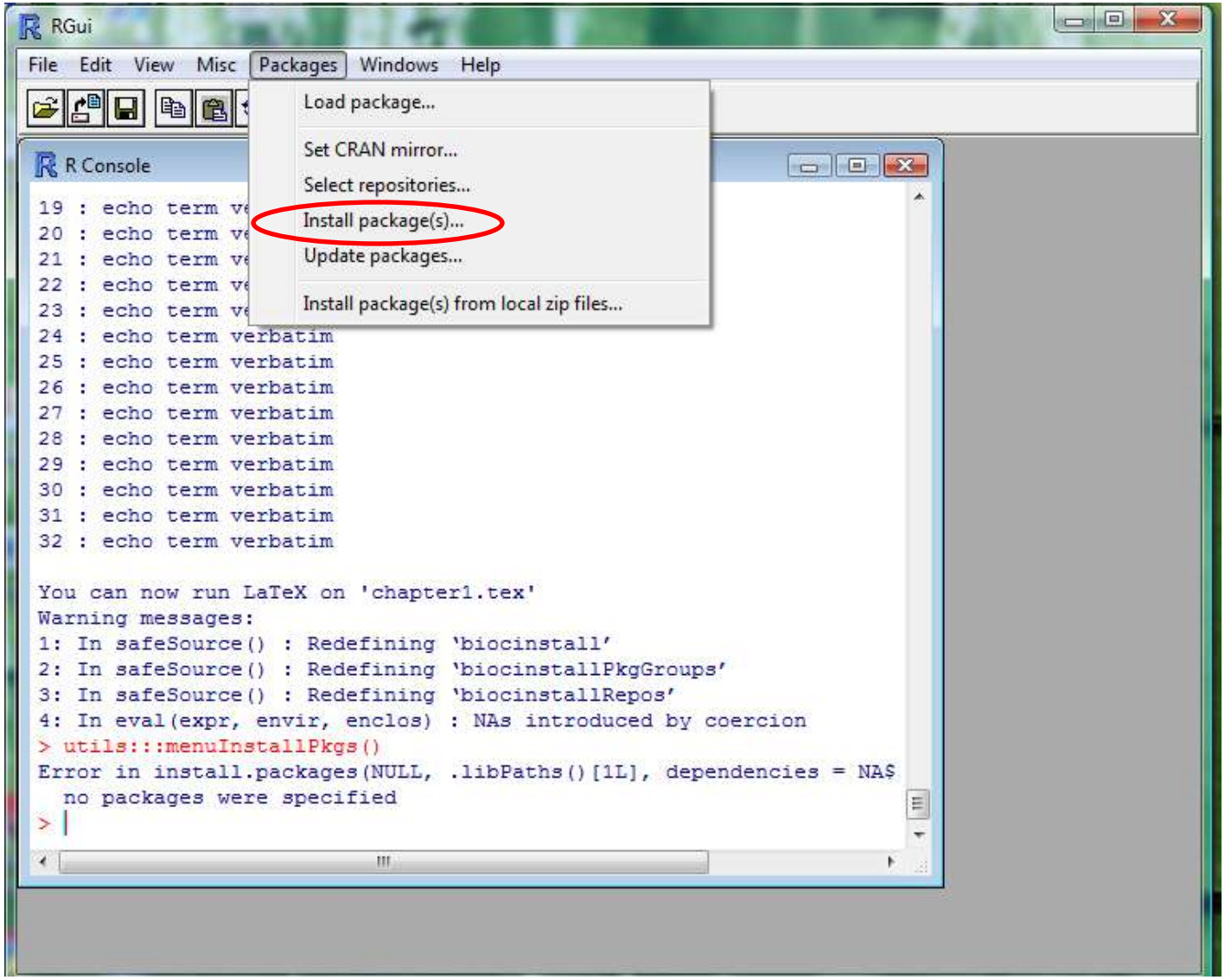
Install Packages from CRAN:

```
> install.packages("Epi")
```

Install Packages from Bioconductor:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("preprocessCore")
```

R System

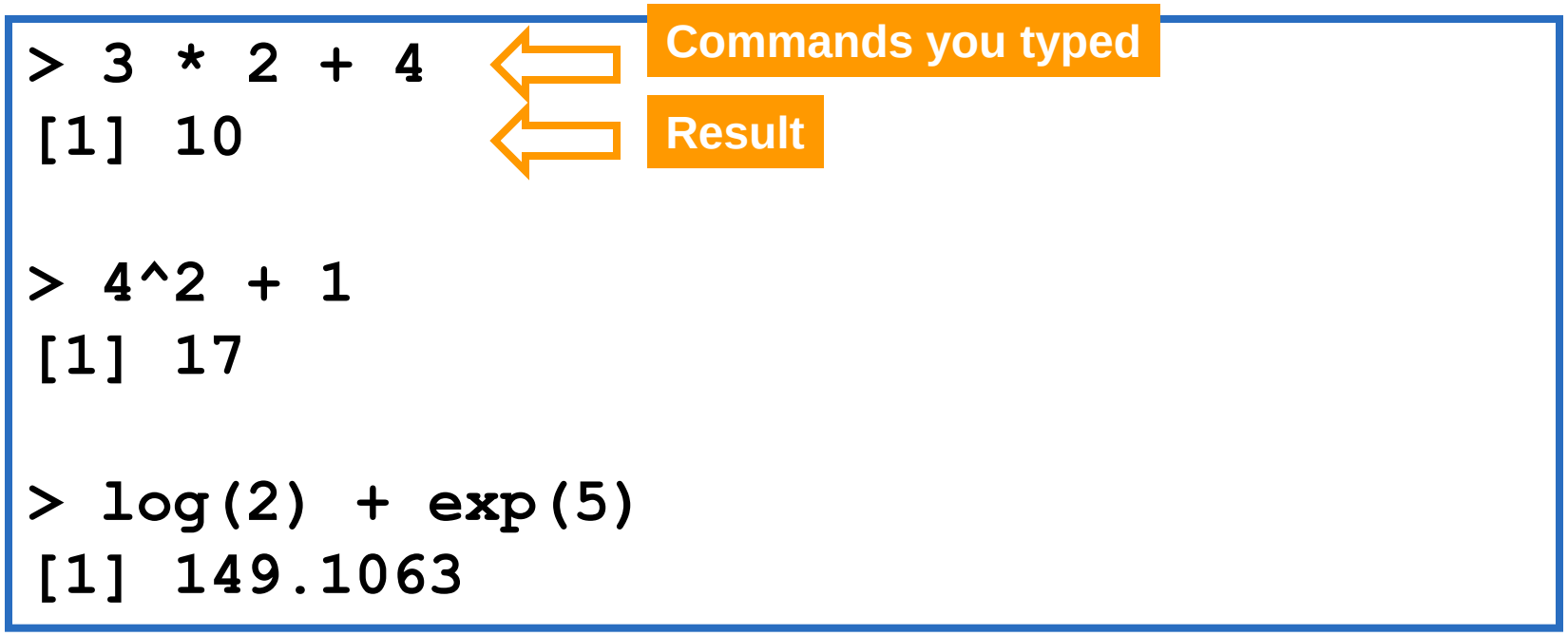

RGui
File Edit View Misc Packages Windows Help
Load package...
Set CRAN mirror...
Select repositories...
Install package(s)...
Update packages...
Install package(s) from local zip files...
R Console
19 : echo term ve
20 : echo term ve
21 : echo term ve
22 : echo term ve
23 : echo term ve
24 : echo term verbatim
25 : echo term verbatim
26 : echo term verbatim
27 : echo term verbatim
28 : echo term verbatim
29 : echo term verbatim
30 : echo term verbatim
31 : echo term verbatim
32 : echo term verbatim
You can now run LaTeX on 'chapter1.tex'
Warning messages:
1: In safeSource() : Redefining 'biocinstall'
2: In safeSource() : Redefining 'biocinstallPkgGroups'
3: In safeSource() : Redefining 'biocinstallRepos'
4: In eval(expr, envir, enclos) : NAs introduced by coercion
> utils:::menuInstallPkgs()
Error in install.packages(NULL, .libPaths()[1L], dependencies = NA$
no packages were specified
>

R System

- Once a package is installed, it is stored on the computer's hard drive
- In order to use a function from the package, you need to use the library function

```
> library(Epi)
```

An Overview of R
Use R as a Calculator



Expressions and Assignments

- Commands in R are
  - expressions
  - assignments
- Commands are separated by
  - a semi-colon
  - a new line
- An expression command is evaluated and (normally) printed

```
> 5 + 2
[1] 7
```

Expressions and Assignments

- You can assign a value to a variable by using **<-** or **=**, but not **< -**

```
> a <- 5 * pi
> a = 5 * pi
> a
[1] 15.70796
> print(a)
[1] 15.70796
```

Naming Conventions

❖ Made up from…

UPPER and/or lower case letters	e.g. **`apple, Apple, APPLE`**
Digits: 0 – 9	e.g. **`People8`**, ~~**`8people`**~~
Periods and underscore:	e.g. **`my.object, my_object`**

🕓 R names are …

case sensitive	**`Apple`** and **`APPLE`** **are different**

🕓 Avoid …	🕓 Do not use the reserved words
`c, q, s, t, C, D, F, I, T, diff, length, mean, pi, range, rank, time, tree, var`	~~**`FALSE, Inf, NA, NaN, NULL, TRUE, break, else, for, function, if, in, next, repeat, return and while`**~~

Vectorized Arithmetic

- A data vector is an array of numbers or characters that can be constructed by using `c(...)`

```
> weight = c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72
> height = c(1.75, 1.8, 1.65, 1.9, 1.74, 1.91)
```

- You can perform the calculations with vectors of the same or different length
- The shorter vector is recycled

```
> bmi = weight/height^2
> bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Vectorized Arithmetic

- If the longer vector is not a multiple of the shorter vector, R will issue a warning message:

```
> a = c(1,2,3)
> b = c(1,2)
> a + b
[1] 2 4 4
Warning message:
In a + b : longer object length is not a multiple of
shorter object length
```

Vectorized Arithmetic

- Calculate the mean of the bmi:

```
> sum(height)
[1] 10.75
> length(height)
[1] 6
> sum(height)/length(height)
[1] 1.791667
```

- Or:

```
> mean(height)
[1] 1.791667
```

Standard Procedures

- Suppose you would like to test the mean of the BMI of the above 6 people, equaling 22.5

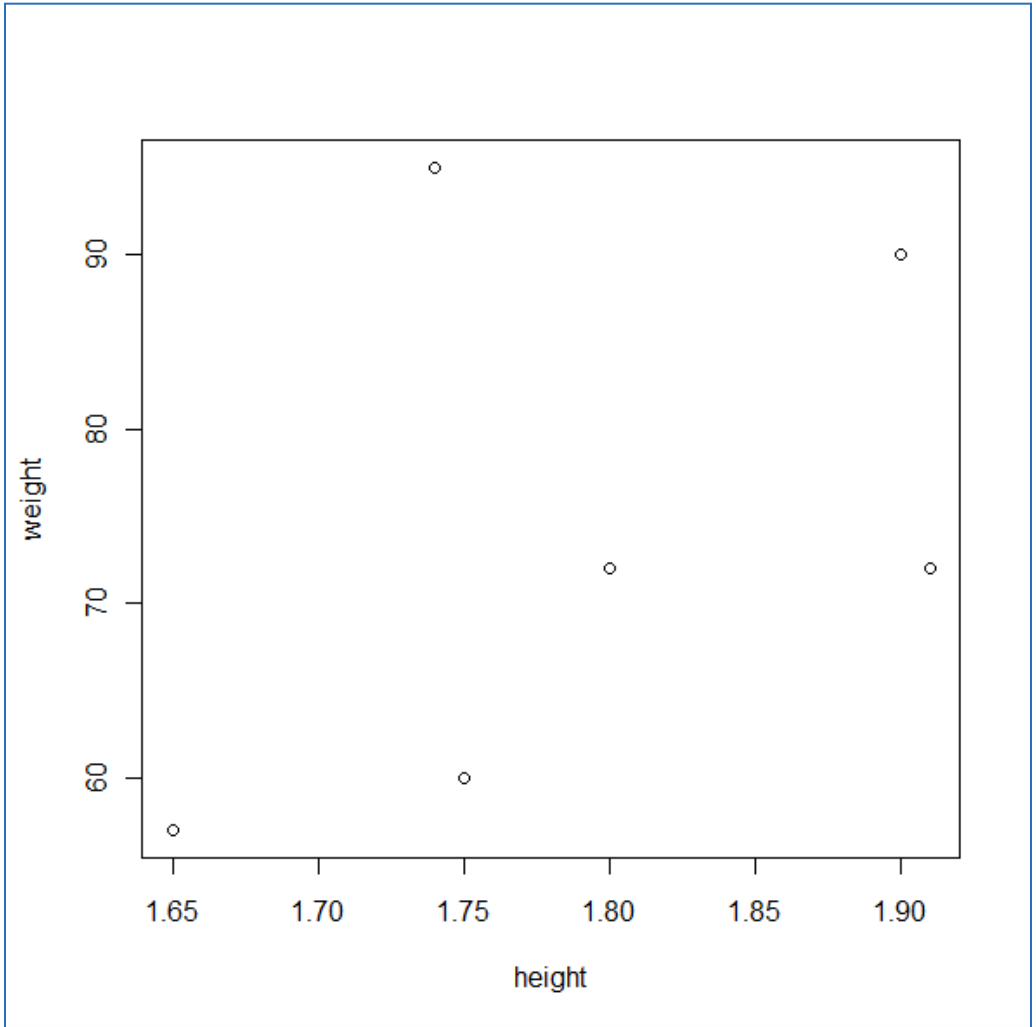
```
> t.test(bmi, mu = 22.5)

        One Sample t-test

data:  bmi
t = 0.3449, df = 5, p-value = 0.7442
alternative hypothesis: true mean is not equal to 22.5
95 percent confidence interval:
 18.41734 27.84791
sample estimates:
mean of x
 23.13262
```

Graphs

```
> plot(height, weight)
```



Miscellaneous Matters

- Everything typed after # be ignored by R
- The previous command can be recalled with the up-arrow key

The `help` Function

```
> ?var
> help(var)
> help(`+`)
```

```
> args(var)
function (x, y = NULL, na.rm = FALSE, use)
NULL
```

```
> help.search("linear models")
```

An Overview of R Objects
Mode and Class

- Everything within the R language is an object
- The nature of R objects can be described by their own attributes.
- The number of attributes for each object vary
- When managing objects in R , it is critical and important to distinguish different R objects and to understand the difference between each object
- The two most important attributes of R objects are mode and class, which describe the characteristics of the objects

Mode and Class

```
> n1 = c(1, 4, pi, 10)
> n1
[1] 1.000000 4.000000 3.141593 10.000000
> mode(n1)
[1] "numeric"
```

- The most commonly encountered modes are
  - numeric
  - character
  - logical

Mode and Class

- The "class" attribute describes the data structure of an object
- The concept of class is closely related to objected-oriented programming

```
> n2 = matrix(n1, nrow = 2)
> n2
     [,1]       [,2]
[1,]    1   3.141593
[2,]    4  10.000000
> mode(n2)
[1] "numeric"
> class(n2)
[1] "matrix"
> class(n1)
[1] "numeric"
```

The `attributes` Function

- Another useful function that can be used to distinguish objects is the **`attributes`** function

```
> attributes(n1)
NULL
> attributes(n2)
$dim
[1] 2 2
```

The `attributes` Function

- To change matrix to a vector, you can also use the **`attributes`** function

```
> n3 = n2
> n3
     [,1]       [,2]
[1,]    1  3.141593
[2,]    4 10.000000
> attributes(n3)
$dim
[1] 2 2
> attributes(n3) = NULL
> n3
[1] 1.000000 4.000000 3.141593 10.000000
```

The `attributes` Function

```
> dim(n2)
[1] 2 2
> dim(n1)
NULL
```

The `is.xxx` Function

- The **`is.xxx`** function tests whether or not its argument is of a required mode or class
- The result from the **`is.xxx`** function is either TRUE or FALSE

```
> is.numeric(n2)
[1] TRUE
> is.character(n2)
[1] FALSE
> is.vector(n2)
[1] FALSE
> is.matrix(n2)
[1] TRUE
```

The `as.xxx` Function

- **`as.xxx`** converts one type of object to another type with different mode or data structure

```
> as.character(n1)
[1] "1"      "4"    "3.14159265358979"   "10"
> as.vector(n2)
[1] 1.000000 4.000000 3.141593 10.000000
```

Missing Values

- **NA** is used for missing elements in numeric, character, and logical vectors
- **NA** is not a value but a marker for a missing value
- The missing values are sometimes intended or may result from computation or data type conversion

```
> c1 = c("A", "B", "C")
> as.numeric(c1)
[1] NA NA NA
Warning message:
In eval(expr, envir, enclos) : NAs introduced
  by coercion
```

Missing Values

- The **`is.na`** function can be used to indicate which elements of a vector are missing
- The length of the result from the **`is.na`** function is the same as its argument

```
> n4 = c(1, NA, 3)
> is.na(n4)
[1] FALSE TRUE FALSE
```

Session Management
Working Directory and Workspace

- Working directory: the directory in which R will look for and save files
- To see the contents of the working directory, type

```
> dir()
[1] "add.header.rownames.txt" "add.header.txt"
[3] "chapter1-009.eps"        "chapter1-009.pdf"
[5] "chapter1-013.eps"        "chapter1-013.pdf"
[7] "chapter1.aux"            "chapter1.bbl"
[9] "chapter1.blg"            "chapter1.idx"
[11] "chapter1.ilg"           "chapter1.ind"
[13] "chapter1.log"           "chapter1.pdf"
[15] "chapter1.ppt"           "chapter1.Rnw"
[17] "chapter1.Rnw.bak"       "chapter1.Rnw.sav"
[19] "chapter1.tex"           "chapter1.tex.bak"
[21] "chapter1.tex.sav"       "preamble.aux"
[23] "preamble.tex"           "preamble.tex.bak"
[25] "sep.tab.txt"            "Sweave.sty"
[27] "template_chapter1.R"
```

Working Directory and Workspace

- The default choice of a working directory, usually an R installation directory, is not a good choice for storing data
- You can change the directory by using the **`setwd()`** function

Working Directory and Workspace

- The workspace is the collection of R objects that are listed upon typing **`ls()`** or **`objects()`**
- All objects created in R are stored in a common workspace

```
> objects()
 [1] "a"                      "biocinstall"
 [3] "biocinstallPkgGroups"   "biocinstallRepos"
 [5] "biocLite"               "bmi"
 [7] "c1"                     "getBioC"
 [9] "height"                 "n1"
[11] "n2"                     "n3"
[13] "n4"                     "sourceBiocinstallScript"
[15] "weight"
```

Working Directory and Workspace

- If you decide to remove some of the objects from the workspace, for example, “height” and “weight”, you can use the **rm** function

```
> rm(height, weight)
> ls()
 [1] "a"                          "biocinstall"
 [3] "biocinstallPkgGroups"       "biocinstallRepos"
 [5] "biocLite"                   "bmi"
 [7] "c1"                         "getBioC"
 [9] "n1"                         "n2"
[11] "n3"                         "n4"
[13] "sourceBiocinstallScript"
```

```
> rm(list=ls())
```

Search Path

- The other places objects most users will care about are those provided in libraries attached by calls to **`library`** or always attached at startup. This comes after the `default place' in the search path
- When R looks for an object named on the command line, it searches through a sequence of places known as the search path

Search Path

```
> library(MASS)
> search()
 [1] ".GlobalEnv"     "package:MASS"      "package:Epi"
 [4] "package:stats"  "package:graphics"  "package:grDevices"
 [7] "package:utils"  "package:datasets"  "package:methods"
[10] "Autoloads"      "package:base"
```

```
> searchpaths()
 [1] ".GlobalEnv"
 [2] "C:/PROGRA~1/R/R-212~1.0/library/MASS"
 [3] "C:/Users/Arthur/Documents/R/win-library/2.12/Epi"
 [4] "C:/PROGRA~1/R/R-212~1.0/library/stats"
 [5] "C:/PROGRA~1/R/R-212~1.0/library/graphics"
 [6] "C:/PROGRA~1/R/R-212~1.0/library/grDevices"
 [7] "C:/PROGRA~1/R/R-212~1.0/library/utils"
 [8] "C:/PROGRA~1/R/R-212~1.0/library/datasets"
 [9] "C:/PROGRA~1/R/R-212~1.0/library/methods"
[10] "Autoloads"
[11] "C:/PROGRA~1/R/R-212~1.0/library/base"
```

Search Path

- The names of the objects held in any place in the search path can be displayed by giving the **`objects`** function

```
> objects(2)
 [1] "abbey"          "accdeaths"     "addterm"
 [4] "Aids2"          "Animals"       "anorexia"
 [7] "area"           "as.fractions"  "bacteria"
[10] "bandwidth.nrd"  "bcv"           "beav1"
[13] "beav2"          "biopsy"        "birthwt"
[16] "Boston"         "boxcox"        "cabbages"
[19] "caith"          "Cars93"        "cats"
[22] "cement"         "chem"          "con2tr"
[25] "contr.sdif"     "coop"          "corresp"
...
```

Search Path

- The function **`find`** can be used to discovers where an object appears on the search path. For example,

```
> find("iris3")
[1] "package:datasets"
> find("quine")
[1] "package:MASS"
```

Search Path

- There is one object in the MASS package named “Traffic”
- Suppose you create an object with the same name in the workspace. Then try to find the “Traffic”

```
> Traffic = c(1, 2, 3)
> find("Traffic")
[1] ".GlobalEnv" "package:MASS"
```

Search Path

- Since the newly created Traffic is in position 1, which is before the position where the MASS package was located, when you type Traffic, you will get the one in position 1

```
> Traffic
[1] 1 2 3
```

```
> get("Traffic", pos = 2)
  year day limit  y
1 1961   1    no 9
2 1961   2    no 11
3 1961   3    no 9
4 1961   4    no 20
5 1961   5    no 31
...
```

```
> conflicts()
[1] "Traffic" "as.Date.numeric" "body<-" "merge.data.frame"
```

Saving Objects

- You can save the workspace to a file at any time. If you type **`save.image()`**, then the objects from the workspace will be saved to the file called **`.Rdata`** in your working directory

- You can also specify a file name, such as **`save.image("lect1.RData")`**

- If you want to save only a few selected objects, you can use the **`save`** function. For example, to save objects, you can type **`save(height, weight, file="ht.RData")`**

- The **`.RData`** file can be loaded by default if you double click on this file. The saved file can be loaded to your current workspace by using the load function. For example, **`load("ht.RData")`**