

This document describes some of the basic functionality of R. Usually, you interact with R through a command-line interface — you type in a command and R responds. Once you have mastered a few commands, the command-line interface gives you efficient control of an extremely powerful tool for interacting with data. Gaining mastery of a few R commands does take some learning and patience as R is finicky. You will undoubtedly experience some challenges as you work to learn a new skill that is not wholly intuitive, but it is well worth the effort. Onward!

The following commands give a glimpse of what R can do. You will find it most beneficial, actually, to type these commands into R to see the results yourself.

Calculating with numbers.

```
> 2 + 2
[1] 4
> 12 * 3 - 10/2 + sqrt(16)
[1] 35
> 3^2
[1] 9
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> sum(1:10)
[1] 55
> mean(1:10)
[1] 5.5
> sd(1:10)
[1] 3.027650
```

You can use R like a calculator. The `*` symbol stands for multiplication and the `^` symbol stands for exponentiation. The colon operator `:` creates an array of numbers from the first to the second. R has a number of built-in functions such as `mean` and `sum` that have obvious meaning. The command `sum(1:10)` calculated

$$1 + 2 + 3 + \cdots + 10$$

The symbol `[1]` that precedes the output says that the row begins with the first number of the output. Long answers are broken across rows, like in this example with similarly useful row labels.

```
> 1:100
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
[19] 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
[55] 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
[91] 91 92 93 94 95 96 97 98 99 100
```

Calculating with arrays. R can do arithmetic operations on arrays. If you multiply an array of numbers by a single number, the multiplication happens separately for each number. You can also add or multiply equal-sized arrays of numbers.

```
> 2 * (1:15)
[1] 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30
> (1:10) + (10:1)
[1] 11 11 11 11 11 11 11 11 11 11
> (1:4)^2
[1] 1  4  9 16
```

Assigning variables. You can use the = sign to create new variables. Typing the name of a variable displays it.

```
> a = 1:10
> a
[1] 1  2  3  4  5  6  7  8  9 10
> mean(a)
[1] 5.5
```

(An alternative to the = syntax is to use the key combination <- which was created to look like an arrow. Older documentation may use this instead of the equal sign, but both are valid methods.)

Combining arrays. The c function in R *concatenates* things. (Because c is a reserved function name in R, it is preferable not to use c as the name of a variable lest you or R gets confused. I tend to use cc when I really want to use c.)

```
> a = 1:10
> b = 15:20
> cc = c(b, a, b)
> cc
[1] 15 16 17 18 19 20  1  2  3  4  5  6  7  8  9 10 15 16 17 18 19 20
```

Using functions. R has many built in functions and you can write your own if you want. To see a function, just type its name. If you want actually *to use* the function, you need to add parentheses at the end, possibly with arguments inbetween.

```
> mean
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
> mean(cc)
[1] 12.04545
```

Here is code to create a function that computes the area of a rectangle.

```
> findArea = function(x, y) {
+   return(x * y)
+ }
> findArea(13, 4)
[1] 52
```

You do not need to type in the `+` prompt. That's how R lets you know that the previous command was incomplete. This can happen when you have a command that begins with a `(` or a `{` and you type a return before the command ends. You can often rectify this by typing the missing `}` or `)`. If you have mistyped and can't get out of the `+` prompt, pressing the **Esc** key usually works to get back to a regular prompt.

Changing your workspace. R keeps all of the variables you keep in memory as it runs. When you end an R session, you may save your workspace. This allows you to have variables you have previously defined available without the need to create them all again from scratch. There will also be times when you will want to read in data sets or read some R code. To do these things more easily, it is often a good idea to change R's working directory.

By default, R will use as its working directory the folder in which the executable program exists. You will most likely want to change the working directory to a new folder where you might keep data from the textbook and your homework. You change the working directory for R under both Windows and Macintosh versions by using the **File** menu and selecting **Change Directory...** with your mouse. My advice is to have a folder where you keep work for this course and to launch R from this folder. If you don't start R from this folder, you can still change the working directory to this folder.

Quitting R. To quit, you can type `q()` on a command line or you can quit through the **File** menu. R will prompt you if you want to save your workspace. Usually, say yes! Say no if, for example, you modified a variable when you didn't mean to, and you don't want this modification to be saved.

Useful shortcuts Try using the "up" and "down" arrows. This will recall previous commands. It is useful when you typed in a long command that included a small typo. You don't have to re-type the whole thing for correcting the typo.

There are several different ways to enter data into R. In the following, we describe several of them.

Entering data using `c`. The easiest way to enter small data sets is with the function `c` that *concatenates* numbers (or vectors) together. For example, we could create an object named ‘glucose’ containing the 31 measures as follows. This data comes from Exercise 2.10 in the third edition of *Statistics for the Life Sciences* by Samuels and Witmer.

```
> glucose = c(81, 85, 93, 93, 99, 76, 75, 84, 78, 84, 81, 82, 89,
+ 81, 96, 82, 74, 70, 84, 86, 80, 70, 131, 75, 88, 102, 115,
+ 89, 82, 79, 106)
```

You do not need to type in the ‘+’ symbols, which are prompts that indicates R is waiting for a command to be completed. As shown here, this command was so long it did not fit onto a single line, so R wrapped to the following line. You could have typed it in without breaking the command into several lines. You may also press [Enter] to continue a command on the next line.

Warning—if you type a ‘(’ and then do not complete the command by typing a ‘)’, R will continue to wait for the command to be completed and show a string of ‘+’ prompts even if you continue to press [Enter]. If you get in trouble, you can press [Esc], the Escape key to break back to a regular prompt.

Entering data using `scan`. The previous example is a bit of a pain, because you need to enter all of the commas. The `scan` function reads in data separated by “white space”, spaces, tabs, and newline characters. If you press [Enter] on a blank line, you end the input. Here is the same example.

```
> glucose = scan()
1: 81 85 93 93 99 76 75 84 78 84 81 82 89 81
15: 96 82 74 70 84 86 80 70 131 75 88 102
27: 115 89 82 79 106
32:
Read 31 items
```

Entering data using `read.table`. Your textbook has a CD with all of the data sets used in the textbook. It is worth learning how to read in these data sets, because it will save you time when working out problems from the book.

You will probably want to begin by creating a folder that will contain the data from the CD. I will assume that you know how to copy the data files from the CD to a folder on your hard drive. There is a folder on the CD called ‘DataFiles’. Under this, there is a folder for each chapter. For each chapter, there are folders for the data sets in different formats. The easiest to read into R are the ASCII formatted files, or plain text files. The data for Exercise 2.10 is in the ‘DataFiles/Chpt 2/ASCII’ folder and is called ‘glucose.txt’.

If you have copied this file to your hard drive *and if you have changed directories through the File menu so that your working directory contains this data file*, this command will read in the data.

```
> x = read.table("glucose.txt", header = TRUE)
```

(If you see an error that the file is not found, it is probably the case that the file is not in your working directory. You may want to try the `file.choose` method described below.)

The file “glucose.txt” is typical for ASCII data sets from the textbook. The first row contains a “header”, the variable names for each variable. In addition to the header, there is one row for each observational unit. This example has only one variable and one column of data. Other examples can have multiple columns of data. The default behavior of R is to read in data frames without headers. That is why it was necessary to add the argument `header=TRUE` to the command. The argument `header=T` would actually be sufficient.

The R commands above create `x`, an object known in R as a *data frame*. The name `x` is arbitrary. You may use any valid variable name (which does not begin with a digit or use characters with other meaning). A data frame is a data matrix, but can include both categorical and numerical variables.

To ensure that you have read in what you think you have, it is useful to use the command `str` to check the structure of the data frame.

```
> str(x)
'data.frame': 31 obs. of 1 variable:
 $ glucose: int  81 85 93 93 99 76 75 84 78 84 ...
```

The output tells you that the data frame `x` has one quantitative variable called `glucose` and it shows the first several values.

After reading in a data frame, R will recognize the name `x` as the name of the entire data frame, but R will not recognize the variable `glucose`. There are two ways to get around this. First, the variables of each data frame may be accessed using the `$` operator. Here is an example.

```
> x$glucose
[1] 81 85 93 93 99 76 75 84 78 84 81 82 89 81 96 82 74 70 84
[20] 86 80 70 131 75 88 102 115 89 82 79 106
```

Alternatively, if you use the `attach` command, you can refer to each variable in `x` by name without the dollar sign.

```
> attach(x)
> glucose
[1] 81 85 93 93 99 76 75 84 78 84 81 82 89 81 96 82 74 70 84
[20] 86 80 70 131 75 88 102 115 89 82 79 106
```

You only need to type in the `attach` command once per session — once you have done it, R will know that `glucose` means the same thing as `x$glucose` until you quit R.

Reading in data using `file.choose`. As an alternative to typing in the file name, you may use the function `file.choose` which will open up a dialog box that you can use to move through your folders to find the data file you want to read in. The syntax is

```
> x = read.table(file.choose(),header=T)
```

Reading in data from an Excel file. R does not read in data directly from an Excel file, but you can use Excel to export data in a format that R can read. To do this, create an Excel spread sheet. Use the first row of the spread sheet as the header row and put the variable values below. Then, save this spread sheet as a CSV file, for comma-separated-variable file. This format is a plain text file, but there will be commas instead of spaces between variable values. Then, in R you would read in the data using `read.table` with the additional argument `sep=","` to indicate that commas are used as separators between fields. When there is only one variable, this is not necessary.

If you have a data file called “students.csv” that looks like this,

```
First,Last,Height,BloodType
Adam,Ant,71.5,A
Jon,Bon Jovi,70,0
Marshal,Mathers,69,B
Elvis,Presley,73,AB
```

you could read in the data and check it with these commands.

```
> students = read.table("students.csv", header = T, sep = ",")
> str(students)
'data.frame': 4 obs. of 4 variables:
 $ First      : Factor w/ 4 levels "Adam","Elvis",...: 1 3 2 4
 $ Last       : Factor w/ 4 levels "Ant","Bon Jovi",...: 1 2 4 3
 $ Height     : num 71.5 70 73 69
 $ BloodType : Factor w/ 4 levels "A","AB","B","O": 1 4 2 3
```