Data Structures

One of R’s most powerful features is its ability to deal with tabular data - such as you may already have in a spreadsheet or a excel file. Let’s start by looking at a wetland dataset in your data/ directory, called WetPlots.xlsx:

## The *readxl* package

To read the data into R, we are going to use our first package, called *readxl*. *readxl* is part of a suite of packages called the “tidyverse” which were designed to work nicely together and to ease many common data operations.

The first time you use a package, you will need to install it (like installing an app on your phone from the app store). Additionally, it is a good idea to periodically check for updates to that package:

install.packages("readxl")

Everytime we want to use that package, you must load it into your R session, by using the library function:

library(readxl)

Now we can load this data into R via the following using the read\_excel function, and assign it to an object called wetland:

wetland <- read\_excel("data/WetPlots.xlsx")

The read\_excel function is used for reading in tabular data from excel There are many other packages that do similar tasks (xlsx, xlsReadWrite), but the readxl version (read\_excel) is a bit more user-friendly, and uses more sensible defaults.

The object that is created by read\_excel is called a “tibble” - a rectangular table-like object with rows and columns that is similar to a data frame.

We can begin exploring our dataset right away, first by looking at the whole thing:

wetland

## # A tibble: 32 x 10  
## fid newid wetland\_id wpt date surveyors slope\_position aspect slope\_pc  
## <chr> <dbl> <chr> <dbl> <chr> <chr> <chr> <chr> <chr>   
## 1 1115~ 40076 (Headwate~ 157 2019~ Don, Dus~ <NA> S 2   
## 2 8798 36819 Morin Lak~ 122 <NA> <NA> <NA> <NA> <NA>   
## 3 8475 36353 Chapman L~ 121 <NA> <NA> <NA> <NA> <NA>   
## 4 1636~ 46206 Kaldo Lake NA 2019~ Carlos, ~ UP NE 3   
## 5 36538 7386 Babine Mo~ 145 2019~ Neil TO S 5   
## 6 88182 31245 Coop Road NA 2019~ LBN, Git~ DH NE 1   
## 7 12519 32714 Fisheries~ NA 2019~ Ken, Law~ <NA> S 1   
## 8 7639 35789 Babine La~ 123 <NA> <NA> <NA> <NA> <NA>   
## 9 44835 6598 Smithers ~ NA 2019~ Jamie, C~ DP NW 2   
## 10 1909~ 36374 Nose #1 142 2019~ <NA> DP <NA> 0   
## # ... with 22 more rows, and 1 more variable: comments <chr>

And pulling out individual columns by specifying them using the $ operator:

wetland$wetland\_id

## [1] "(Headwaters to) Gitanyow Lake" "Morin Lake ID: 13599"   
## [3] "Chapman Lake South (19349)" "Kaldo Lake"   
## [5] "Babine Mountain" "Coop Road"   
## [7] "Fisheries - LBN" "Babine Lake Road (20818)"   
## [9] "Smithers (Canadian Tire)" "Nose #1"   
## [11] "Austion Road Site" "Caribou Rd 1"   
## [13] NA NA   
## [15] "Nass 1 (?)" "Hwy 37 (Plot 7)"   
## [17] "Squish #4" "Squish Plot #3"   
## [19] "Squish #5" "Surprise #6 - Hwy 37A West"   
## [21] "LBN - Squish" "Crow Road"   
## [23] "Grizzzly #1" "Kuldo"   
## [25] "PP1" "PP7"   
## [27] "SS13" "SS1"   
## [29] "Dublisxw01" "Dublisxw02"   
## [31] "Dublisxw04" "Dulisxw03"

wetland$surveyors

## [1] "Don, Dustin, Neil"   
## [2] NA   
## [3] NA   
## [4] "Carlos, Jasmine"   
## [5] "Neil"   
## [6] "LBN, Gitxsan, Don, Neil"   
## [7] "Ken, Lawrence"   
## [8] NA   
## [9] "Jamie, Carlos, Dustin, Greg, Neil"   
## [10] NA   
## [11] NA   
## [12] NA   
## [13] "Lukas (?), Roy Wilson (Xsi Duutsuit) Klispiox (?) 250-842-8224"  
## [14] NA   
## [15] "Carlos, Jamie"   
## [16] "GJ, DG"   
## [17] "GJ, DG"   
## [18] "GJ,DG,DM,NF"   
## [19] "GJ,DG"   
## [20] "GJ, DG"   
## [21] NA   
## [22] "Neil, Jamie, Carlos, Norm, Lawrence, Don"   
## [23] "Lawrence, Norm, Ken, Neil"   
## [24] "Jamie & Carlos"   
## [25] "Jamie and Carlos"   
## [26] "Jamie and Carlos"   
## [27] "Jamie and Carlos"   
## [28] "Jamie and Carlos"   
## [29] "Jamie and Carlos"   
## [30] "Jamie and Carlos"   
## [31] "Jamie and Carlos"   
## [32] "Jamie and Carlos"

wetland$newid

## [1] 40076 36819 36353 46206 7386 31245 32714 35789 6598 36374 36041 36786  
## [13] 45370 43391 43077 43723 41084 43391 40339 48222 42843 32338 32782 47549  
## [25] 41235 41123 42468 41817 7269 7140 7011 7110

We can do other operations on the columns:

## Say we wanted to increase the size of our id number by 10000  
wetland$newid + 10000

## [1] 50076 46819 46353 56206 17386 41245 42714 45789 16598 46374 46041 46786  
## [13] 55370 53391 53077 53723 51084 53391 50339 58222 52843 42338 42782 57549  
## [25] 51235 51123 52468 51817 17269 17140 17011 17110

paste("Site ", wetland$newid, "was surveyed by", wetland$surveyors)

## [1] "Site 40076 was surveyed by Don, Dustin, Neil"   
## [2] "Site 36819 was surveyed by NA"   
## [3] "Site 36353 was surveyed by NA"   
## [4] "Site 46206 was surveyed by Carlos, Jasmine"   
## [5] "Site 7386 was surveyed by Neil"   
## [6] "Site 31245 was surveyed by LBN, Gitxsan, Don, Neil"   
## [7] "Site 32714 was surveyed by Ken, Lawrence"   
## [8] "Site 35789 was surveyed by NA"   
## [9] "Site 6598 was surveyed by Jamie, Carlos, Dustin, Greg, Neil"   
## [10] "Site 36374 was surveyed by NA"   
## [11] "Site 36041 was surveyed by NA"   
## [12] "Site 36786 was surveyed by NA"   
## [13] "Site 45370 was surveyed by Lukas (?), Roy Wilson (Xsi Duutsuit) Klispiox (?) 250-842-8224"  
## [14] "Site 43391 was surveyed by NA"   
## [15] "Site 43077 was surveyed by Carlos, Jamie"   
## [16] "Site 43723 was surveyed by GJ, DG"   
## [17] "Site 41084 was surveyed by GJ, DG"   
## [18] "Site 43391 was surveyed by GJ,DG,DM,NF"   
## [19] "Site 40339 was surveyed by GJ,DG"   
## [20] "Site 48222 was surveyed by GJ, DG"   
## [21] "Site 42843 was surveyed by NA"   
## [22] "Site 32338 was surveyed by Neil, Jamie, Carlos, Norm, Lawrence, Don"   
## [23] "Site 32782 was surveyed by Lawrence, Norm, Ken, Neil"   
## [24] "Site 47549 was surveyed by Jamie & Carlos"   
## [25] "Site 41235 was surveyed by Jamie and Carlos"   
## [26] "Site 41123 was surveyed by Jamie and Carlos"   
## [27] "Site 42468 was surveyed by Jamie and Carlos"   
## [28] "Site 41817 was surveyed by Jamie and Carlos"   
## [29] "Site 7269 was surveyed by Jamie and Carlos"   
## [30] "Site 7140 was surveyed by Jamie and Carlos"   
## [31] "Site 7011 was surveyed by Jamie and Carlos"   
## [32] "Site 7110 was surveyed by Jamie and Carlos"

But what about

wetland$newid + wetland$surveyors

## Error in wetland$newid + wetland$surveyors: non-numeric argument to binary operator

Understanding what happened here is key to successfully analyzing data in R.

## Data Types

If you guessed that the last command will return an error because site 1 with an id of 40076 plus "Don, Dustin, Neil" is nonsense, you’re right - and you already have some intuition for an important concept in programming called *data types*. We can ask what type of data something is:

typeof(wetland$wetland\_id)

## [1] "character"

There are 4 main types:

* double/numeric (decimal numbers),
* integer (counting numbers),
* logical (True/False),
* character (free text)

typeof(3.14)

## [1] "double"

typeof(1L) # The L suffix forces the number to be an integer, since by default R uses double (decimal) numbers

## [1] "integer"

typeof(TRUE)

## [1] "logical"

typeof('banana')

## [1] "character"

No matter how complicated our analyses become, all data in R is interpreted as one of these basic data types. This strictness has some really important consequences.

The table that R loaded our wetland data into is something called a *tibble*, and it is our first example of something called a *data structure* - that is, a structure which R knows how to build out of the basic data types.

class(wetland)

## [1] "tbl\_df" "tbl" "data.frame"

## Vectors and Type Coercion

To better understand this behavior, let’s meet another of the data structures: the *vector*.

If we are creating vectors on our own, we will normally use the c (combine) function:

my\_vector <- c(1, 3, 5 ,7 ,9)  
my\_vector

## [1] 1 3 5 7 9

A vector in R is essentially an ordered list of things, with the special condition that *everything in the vector must be the same basic data type*.

class(my\_vector)

## [1] "numeric"

This command indicates the basic data type found in this vector - in this case numeric.

We can use the logical operators that we learned earlier with vectors:

my\_vector > 4

## [1] FALSE FALSE TRUE TRUE TRUE

Vectors can be any data type that we’ve already learned about. Let’s make a character vector:

my\_other\_vector <- c("Moose", "Bear", "Wolf", "Deer")  
my\_other\_vector

## [1] "Moose" "Bear" "Wolf" "Deer"

class(my\_other\_vector)

## [1] "character"

my\_other\_vector == "Wolf"

## [1] FALSE FALSE TRUE FALSE

If we similarly do

class(wetland$newid)

## [1] "numeric"

we see that class(wetland$newid) is a vector, too - *the columns of data we load into R data.frames are all vectors*, and that’s the root of why R forces everything in a column to be the same basic data type.

## Discussion 1

Why is R so opinionated about what we put in our columns of data? How does this help us?

Given what we’ve learned so far, what do you think the following will produce?

quiz\_vector <- c(2,6,'3')

This is something called *type coercion*, and it is the source of many surprises and the reason why we need to be aware of the basic data types and how R will interpret them. When R encounters a mix of types (here numeric and character) to be combined into a single vector, it will force them all to be the same type. Consider:

coercion\_vector <- c('a', TRUE)  
coercion\_vector

## [1] "a" "TRUE"

another\_coercion\_vector <- c(0, TRUE)  
another\_coercion\_vector

## [1] 0 1

The coercion rules go: logical -> integer -> double/numeric -> complex -> character, where -> can be read as *are transformed into*. You can try to force coercion against this flow using the as. functions:

character\_vector\_example <- c('0','2','4')  
character\_vector\_example

## [1] "0" "2" "4"

character\_coerced\_to\_numeric <- as.numeric(character\_vector\_example)  
character\_coerced\_to\_numeric

## [1] 0 2 4

numeric\_coerced\_to\_logical <- as.logical(character\_coerced\_to\_numeric)  
numeric\_coerced\_to\_logical

## [1] FALSE TRUE TRUE

As you can see, some surprising things can happen when R forces one basic data type into another! Nitty-gritty of type coercion aside, the point is: if your data doesn’t look like what you thought it was going to look like, type coercion may well be to blame; make sure everything is the same type in your vectors and your columns of data.frames, or you will get nasty surprises!

But coercion can also be very useful! For example, in our wetland data slope\_pc is numeric, but we know that it also contains a “flat” which is not numeric. Therefore R will convert all the column into a character type.

wetland$slope\_pc

## [1] "2" NA NA "3" "5" "1" "1" NA "2" "0"   
## [11] "2" "1" "1" NA "2" "flat" NA NA NA NA   
## [21] NA "3" "0" NA NA NA NA NA NA NA   
## [31] NA NA

as.numeric(wetland$slope\_pc)

## Warning: NAs introduced by coercion

## [1] 2 NA NA 3 5 1 1 NA 2 0 2 1 1 NA 2 NA NA NA NA NA NA 3 0 NA NA  
## [26] NA NA NA NA NA NA NA

# note what happened to the "flat" site?

## Data Frames

We said that columns in data.frames were vectors:

wetland$wetland\_id

## [1] "(Headwaters to) Gitanyow Lake" "Morin Lake ID: 13599"   
## [3] "Chapman Lake South (19349)" "Kaldo Lake"   
## [5] "Babine Mountain" "Coop Road"   
## [7] "Fisheries - LBN" "Babine Lake Road (20818)"   
## [9] "Smithers (Canadian Tire)" "Nose #1"   
## [11] "Austion Road Site" "Caribou Rd 1"   
## [13] NA NA   
## [15] "Nass 1 (?)" "Hwy 37 (Plot 7)"   
## [17] "Squish #4" "Squish Plot #3"   
## [19] "Squish #5" "Surprise #6 - Hwy 37A West"   
## [21] "LBN - Squish" "Crow Road"   
## [23] "Grizzzly #1" "Kuldo"   
## [25] "PP1" "PP7"   
## [27] "SS13" "SS1"   
## [29] "Dublisxw01" "Dublisxw02"   
## [31] "Dublisxw04" "Dulisxw03"

wetland$surveyors

## [1] "Don, Dustin, Neil"   
## [2] NA   
## [3] NA   
## [4] "Carlos, Jasmine"   
## [5] "Neil"   
## [6] "LBN, Gitxsan, Don, Neil"   
## [7] "Ken, Lawrence"   
## [8] NA   
## [9] "Jamie, Carlos, Dustin, Greg, Neil"   
## [10] NA   
## [11] NA   
## [12] NA   
## [13] "Lukas (?), Roy Wilson (Xsi Duutsuit) Klispiox (?) 250-842-8224"  
## [14] NA   
## [15] "Carlos, Jamie"   
## [16] "GJ, DG"   
## [17] "GJ, DG"   
## [18] "GJ,DG,DM,NF"   
## [19] "GJ,DG"   
## [20] "GJ, DG"   
## [21] NA   
## [22] "Neil, Jamie, Carlos, Norm, Lawrence, Don"   
## [23] "Lawrence, Norm, Ken, Neil"   
## [24] "Jamie & Carlos"   
## [25] "Jamie and Carlos"   
## [26] "Jamie and Carlos"   
## [27] "Jamie and Carlos"   
## [28] "Jamie and Carlos"   
## [29] "Jamie and Carlos"   
## [30] "Jamie and Carlos"   
## [31] "Jamie and Carlos"   
## [32] "Jamie and Carlos"

## Factors

Another important data structure is called a *factor*. Factors usually look like character data, but are typically used to represent categorical information that have a defined set of values. For example, let’s make a vector of strings labelling the aspect in our study. This can only every be one of 8 catergories (N, NE, E, SE, S, SW, W, NW) and missing value (NA):

aspect <- c('N', 'NE', 'E', 'SE', 'S', 'SW', 'W', 'NW')  
aspect

## [1] "N" "NE" "E" "SE" "S" "SW" "W" "NW"

We can turn a vector into a factor like so:

aspect\_class <- factor(aspect)  
class(aspect\_class)

## [1] "factor"

aspect\_class

## [1] N NE E SE S SW W NW  
## Levels: E N NE NW S SE SW W

## Factors - bonus details

In modelling functions, it’s important to know what the baseline levels are. This is assumed to be the first factor, but by default factors are labelled in alphabetical order. You can change this by specifying the levels:

mydata <- c("case", "control", "control", "case")  
factor\_ordering\_example <- factor(mydata, levels = c("control", "case"))  
str(factor\_ordering\_example)

## Factor w/ 2 levels "control","case": 2 1 1 2

Now R has noticed that there are two possible categories in our data - but it also did something surprising; instead of printing out the strings we gave it, we got a bunch of numbers instead. R has replaced our human-readable categories with numbered indices under the hood, this is necessary as many statistical calculations utilise such numerical representations for categorical data.

In this case, we’ve explicitly told R that “control” should be represented by 1, and “case” by 2. This designation can be very important for interpreting the results of statistical models!

## Lists

Another data structure you’ll want in your bag of tricks is the list. A list is simpler in some ways than the other types, because you can put anything you want in it:

list\_example <- list(1, "a", TRUE, 1+4i)  
list\_example

## [[1]]  
## [1] 1  
##   
## [[2]]  
## [1] "a"  
##   
## [[3]]  
## [1] TRUE  
##   
## [[4]]  
## [1] 1+4i

another\_list <- list(title = "Numbers", numbers = 1:10, data = TRUE )  
another\_list

## $title  
## [1] "Numbers"  
##   
## $numbers  
## [1] 1 2 3 4 5 6 7 8 9 10  
##   
## $data  
## [1] TRUE

We can now understand something a bit surprising in our data.frame; what happens if we run:

typeof(wetland)

## [1] "list"

We see that data.frames look like lists ‘under the hood’ - this is because a data.frame is really a list of vectors and factors, as they have to be - in order to hold those columns that are a mix of vectors and factors, the data.frame needs something a bit more flexible than a vector to put all the columns together into a familiar table. In other words, a data.frame is a special list in which all the vectors must have the same length.

In our wetland example, we have character and numeric variables we have seen already, each column of data.frame is a vector.

wetland$newid

## [1] 40076 36819 36353 46206 7386 31245 32714 35789 6598 36374 36041 36786  
## [13] 45370 43391 43077 43723 41084 43391 40339 48222 42843 32338 32782 47549  
## [25] 41235 41123 42468 41817 7269 7140 7011 7110

wetland[,2]

## # A tibble: 32 x 1  
## newid  
## <dbl>  
## 1 40076  
## 2 36819  
## 3 36353  
## 4 46206  
## 5 7386  
## 6 31245  
## 7 32714  
## 8 35789  
## 9 6598  
## 10 36374  
## # ... with 22 more rows

str(wetland[,2])

## Classes 'tbl\_df', 'tbl' and 'data.frame': 32 obs. of 1 variable:  
## $ newid: num 40076 36819 36353 46206 7386 ...

Each row is an *observation* of different variables, itself a data.frame, and thus can be composed of elements of different types.

wetland[1,]

## # A tibble: 1 x 10  
## fid newid wetland\_id wpt date surveyors slope\_position aspect slope\_pc  
## <chr> <dbl> <chr> <dbl> <chr> <chr> <chr> <chr> <chr>   
## 1 1115~ 40076 (Headwate~ 157 2019~ Don, Dus~ <NA> S 2   
## # ... with 1 more variable: comments <chr>

typeof(wetland[1,])

## [1] "list"

str(wetland[1,])

## Classes 'tbl\_df', 'tbl' and 'data.frame': 1 obs. of 10 variables:  
## $ fid : chr "111563"  
## $ newid : num 40076  
## $ wetland\_id : chr "(Headwaters to) Gitanyow Lake"  
## $ wpt : num 157  
## $ date : chr "2019-08-30"  
## $ surveyors : chr "Don, Dustin, Neil"  
## $ slope\_position: chr NA  
## $ aspect : chr "S"  
## $ slope\_pc : chr "2"  
## $ comments : chr "157"

## Challenge 1

There are several subtly different ways to call variables, observations and elements from data.frames:

* wetland[1]
* wetland[[1]]
* wetland$fid
* wetland["fid"]
* wetland[1, 1]
* wetland[, 1]
* wetland[1, ]

Try out these examples and explain what is returned by each one.

*Hint:* Use the function typeof() to examine what is returned in each case.

## Matrices

Last but not least is the matrix. We can declare a matrix full of zeros:

matrix\_example <- matrix(0, ncol=6, nrow=3)  
matrix\_example

## [,1] [,2] [,3] [,4] [,5] [,6]  
## [1,] 0 0 0 0 0 0  
## [2,] 0 0 0 0 0 0  
## [3,] 0 0 0 0 0 0

And similar to other data structures, we can ask things about our matrix:

class(matrix\_example)

## [1] "matrix"

typeof(matrix\_example)

## [1] "double"

str(matrix\_example)

## num [1:3, 1:6] 0 0 0 0 0 0 0 0 0 0 ...

dim(matrix\_example)

## [1] 3 6

nrow(matrix\_example)

## [1] 3

ncol(matrix\_example)

## [1] 6

## Challenge 2

What do you think will be the result of length(matrix\_example)? Try it. Were you right? Why / why not?

## Challenge 3

Make another matrix, this time containing the numbers 1:50, with 5 columns and 10 rows. Did the matrix function fill your matrix by column, or by row, as its default behaviour? See if you can figure out how to change this. (hint: read the documentation for matrix!)

## Challenge 4

Consider the R output of the matrix below:

## [,1] [,2]  
## [1,] 4 1  
## [2,] 9 5  
## [3,] 10 7

What was the correct command used to write this matrix? Examine each command and try to figure out the correct one before typing them. Think about what matrices the other commands will produce.

1. matrix(c(4, 1, 9, 5, 10, 7), nrow = 3)
2. matrix(c(4, 9, 10, 1, 5, 7), ncol = 2, byrow = TRUE)
3. matrix(c(4, 9, 10, 1, 5, 7), nrow = 2)
4. matrix(c(4, 1, 9, 5, 10, 7), ncol = 2, byrow = TRUE)

## Discussion 1

By keeping everything in a column the same, we allow ourselves to make simple assumptions about our data; if you can interpret one entry in the column as a number, then you can interpret *all* of them as numbers, so we don’t have to check every time. This consistency is what people mean when they talk about *clean data*; in the long run, strict consistency goes a long way to making our lives easier in R.

## Solution to Challenge 1

wetland[1]

## # A tibble: 32 x 1  
## fid   
## <chr>   
## 1 111563  
## 2 8798   
## 3 8475   
## 4 163604  
## 5 36538   
## 6 88182   
## 7 12519   
## 8 7639   
## 9 44835   
## 10 190975  
## # ... with 22 more rows

We can think of a data frame as a list of vectors. The single brace [1] returns the first slice of the list, as another list. In this case it is the first column of the data frame.

wetland[[1]]

## [1] "111563" "8798" "8475" "163604" "36538" "88182" "12519"   
## [8] "7639" "44835" "190975" "236997" "191640" "166151" "109507"   
## [15] "109411" "171712" "109171" "109507b" "139285" "111036" "109454"   
## [22] "14539" "19670" "163828" "...26" "...27" "...28" "...29"   
## [29] "...30" "...31" "...32" "...33"

The double brace [[1]] returns the contents of the list item. In this case it is the contents of the first column, a *vector* of type *factor*.

wetland$fid

## [1] "111563" "8798" "8475" "163604" "36538" "88182" "12519"   
## [8] "7639" "44835" "190975" "236997" "191640" "166151" "109507"   
## [15] "109411" "171712" "109171" "109507b" "139285" "111036" "109454"   
## [22] "14539" "19670" "163828" "...26" "...27" "...28" "...29"   
## [29] "...30" "...31" "...32" "...33"

This example uses the $ character to address items by name. *fid* is the first column of the data frame, again a *vector* of type *factor*.

wetland["fid"]

## # A tibble: 32 x 1  
## fid   
## <chr>   
## 1 111563  
## 2 8798   
## 3 8475   
## 4 163604  
## 5 36538   
## 6 88182   
## 7 12519   
## 8 7639   
## 9 44835   
## 10 190975  
## # ... with 22 more rows

Here we are using a single brace ["fid"] replacing the index number with the column name. Like example 1, the returned object is a *list*.

wetland[1, 1]

## # A tibble: 1 x 1  
## fid   
## <chr>   
## 1 111563

This example uses a single brace, but this time we provide row and column coordinates. The returned object is the value in row 1, column 1. The object is an *integer* but because it is part of a *vector* of type *factor*, R displays the label “calico” associated with the integer value.

wetland[, 1]

## # A tibble: 32 x 1  
## fid   
## <chr>   
## 1 111563  
## 2 8798   
## 3 8475   
## 4 163604  
## 5 36538   
## 6 88182   
## 7 12519   
## 8 7639   
## 9 44835   
## 10 190975  
## # ... with 22 more rows

Like the previous example we use single braces and provide row and column coordinates. The row coordinate is not specified, R interprets this missing value as all the elements in this *column* *vector*.

wetland[1, ]

## # A tibble: 1 x 10  
## fid newid wetland\_id wpt date surveyors slope\_position aspect slope\_pc  
## <chr> <dbl> <chr> <dbl> <chr> <chr> <chr> <chr> <chr>   
## 1 1115~ 40076 (Headwate~ 157 2019~ Don, Dus~ <NA> S 2   
## # ... with 1 more variable: comments <chr>

Again we use the single brace with row and column coordinates. The column coordinate is not specified. The return value is a *list* containing all the values in the first row.

## Solution to Challenge 2

What do you think will be the result of length(matrix\_example)?

matrix\_example <- matrix(0, ncol=6, nrow=3)  
 length(matrix\_example)

## [1] 18

Because a matrix is a vector with added dimension attributes, length gives you the total number of elements in the matrix.

## Solution to Challenge 3

Make another matrix, this time containing the numbers 1:50, with 5 columns and 10 rows. Did the matrix function fill your matrix by column, or by row, as its default behaviour? See if you can figure out how to change this. (hint: read the documentation for matrix!)

x <- matrix(1:50, ncol=5, nrow=10)  
 x <- matrix(1:50, ncol=5, nrow=10, byrow = TRUE) # to fill by row

## Solution to Challenge 4

Consider the R output of the matrix below:

## [,1] [,2]  
## [1,] 4 1  
## [2,] 9 5  
## [3,] 10 7

What was the correct command used to write this matrix? Examine each command and try to figure out the correct one before typing them. Think about what matrices the other commands will produce.

matrix(c(4, 1, 9, 5, 10, 7), ncol = 2, byrow = TRUE)