University of BRISTOL

Getting started with R

A beginner’s guide to R and R Studio

Henry W J Reeve

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.

Statistical Computing & Empirical Methods

What will we cover today?

- We will introduce R and RStudio
- We will learn about the basic objects and operations
- We will learn how to write a simple function in R with control flow statements
- We will see how R facilitates a functional paradigm with call-by-value semantics
- We have a brief look at lazy evaluation
- We will give a few signposts for where to learn more.

Why R?

- R is a free, open source programming language designed for Statistical Computing.
- R provides a fantastic ecosystem for:
  a) Graphics and data visualization
  b) Efficient data wrangling
  c) Statistical inference
  d) Machine Learning
- Vast online community of contributors and enthusiasts!
- Remark: Python & Julia provide increasingly competitive alternatives.

Why RStudio?

- RStudio is a free, open source *integrated development environment* for R.
- RStudio provides:

a) A console with a command line interface

b) A source editor for writing, executing and debugging R scripts

c) Syntax highlighting, code completion, smart indentation

d) Reproducible analysis via knitr & R Markdown.

e) Convenient interface for version control via Git

Installing R and RStudio

- You can install both R and RStudio in Windows, Linux or Mac OS X.
- First download and install R from the Comprehensive R Archive Network:

http://www.r-project.org

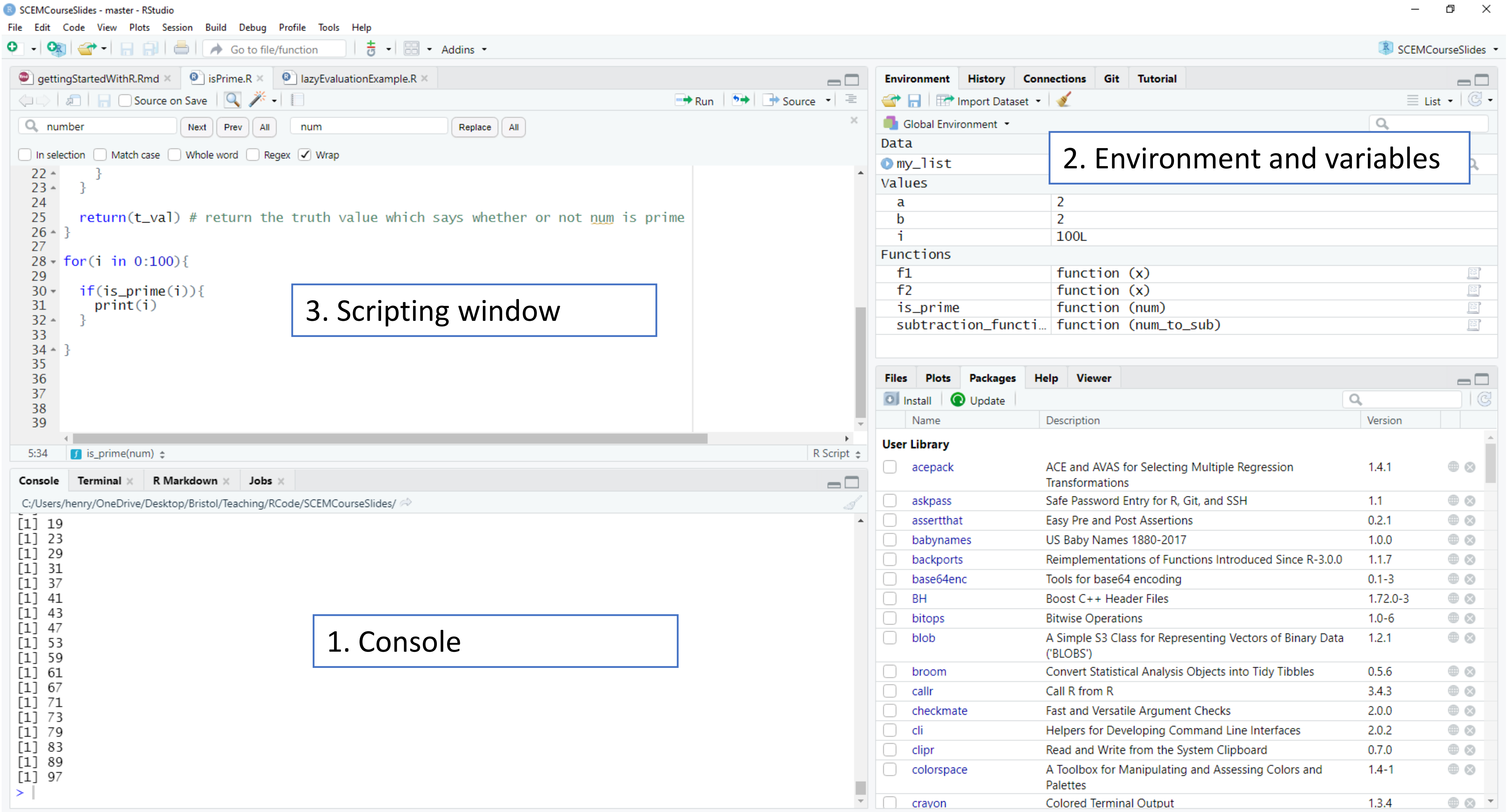
- Then download and install Rstudio:

http://www.rstudio.com/download

- You can also find the links within the assignment!

Now take a break!

Your first R session


SCEMCourseSlides - master - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
SCEMCourseSlides
gettingStartedWithR.Rmd
isPrime.R
lazyEvaluationExample.R
Source on Save
Run
Source
Next
Prev
All
Replace
All
Environment History Connections Git Tutorial
Import Dataset
List
Global Environment
Console Terminal R Markdown Jobs
Files Plots Packages Help Viewer
Install
Update
R Script

Find: number — Replace: num

□ In selection □ Match case □ Whole word □ Regex ☑ Wrap

```
    }
  }

  return(t_val) # return the truth value which says whether or not num is prime
}

for(i in 0:100){

  if(is_prime(i)){
    print(i)
  }

}
```

5:34 is_prime(num)

C:/Users/henry/OneDrive/Desktop/Bristol/Teaching/RCode/SCEMCourseSlides/

```
[1] 19
[1] 23
[1] 29
[1] 31
[1] 37
[1] 41
[1] 43
[1] 47
[1] 53
[1] 59
[1] 61
[1] 67
[1] 71
[1] 73
[1] 79
[1] 83
[1] 89
[1] 97
>
```

Data	
my_list	
Values	
a	2
b	2
i	100L
Functions	
f1	function (x)
f2	function (x)
is_prime	function (num)
subtraction_functi…	function (num_to_sub)

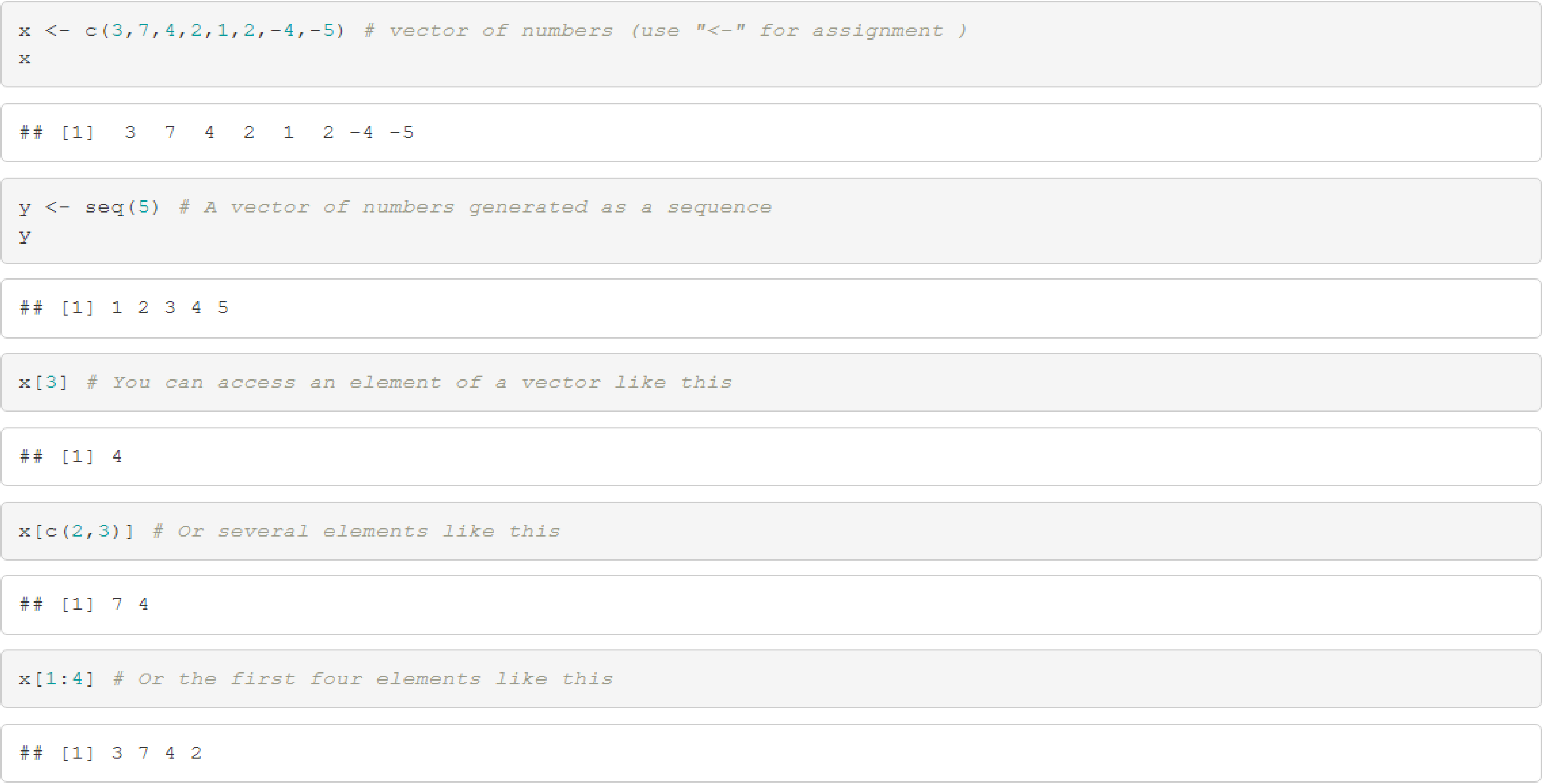
Name	Description	Version
User Library		
□ acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
□ askpass	Safe Password Entry for R, Git, and SSH	1.1
□ assertthat	Easy Pre and Post Assertions	0.2.1
□ babynames	US Baby Names 1880-2017	1.0.0
□ backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.7
□ base64enc	Tools for base64 encoding	0.1-3
□ BH	Boost C++ Header Files	1.72.0-3
□ bitops	Bitwise Operations	1.0-6
□ blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.1
□ broom	Convert Statistical Analysis Objects into Tidy Tibbles	0.5.6
□ callr	Call R from R	3.4.3
□ checkmate	Fast and Versatile Argument Checks	2.0.0
□ cli	Helpers for Developing Command Line Interfaces	2.0.2
□ clipr	Read and Write from the System Clipboard	0.7.0
□ colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	1.4-1
□ crayon	Colored Terminal Output	1.3.4

2. Environment and variables

3. Scripting window

1. Console

Vectors



```
x <- c(3,7,4,2,1,2,-4,-5) # vector of numbers (use "<-" for assignment )
x
```

```
## [1]  3  7  4  2  1  2 -4 -5
```

```
y <- seq(5) # A vector of numbers generated as a sequence
y
```

```
## [1] 1 2 3 4 5
```

```
x[3] # You can access an element of a vector like this
```

```
## [1] 4
```

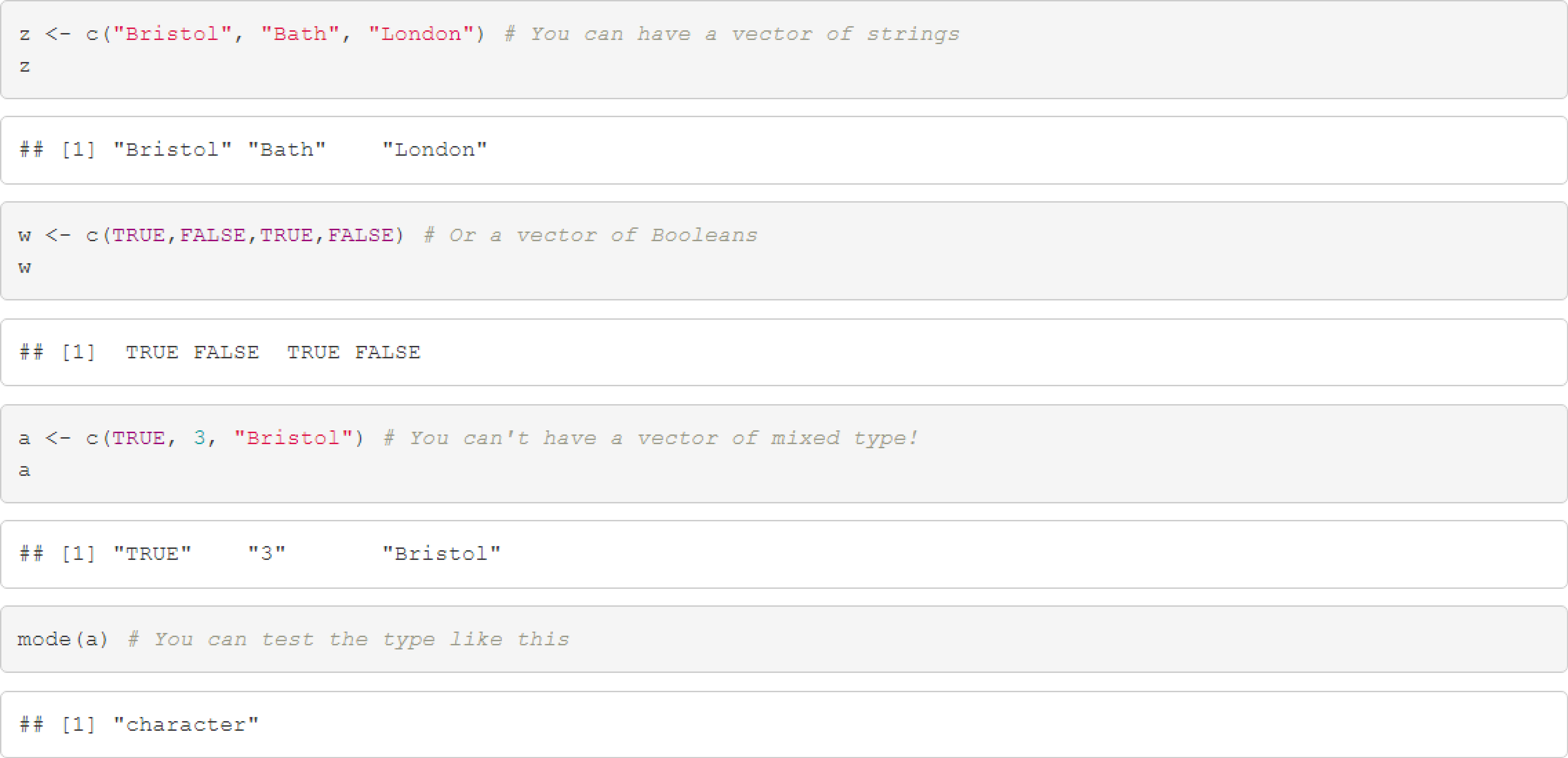
```
x[c(2,3)] # Or several elements like this
```

```
## [1] 7 4
```

```
x[1:4] # Or the first four elements like this
```

```
## [1] 3 7 4 2
```

Vectors



```
z <- c("Bristol", "Bath", "London") # You can have a vector of strings
z
```

```
## [1] "Bristol" "Bath"    "London"
```

```
w <- c(TRUE,FALSE,TRUE,FALSE) # Or a vector of Booleans
w
```

```
## [1]  TRUE FALSE  TRUE FALSE
```

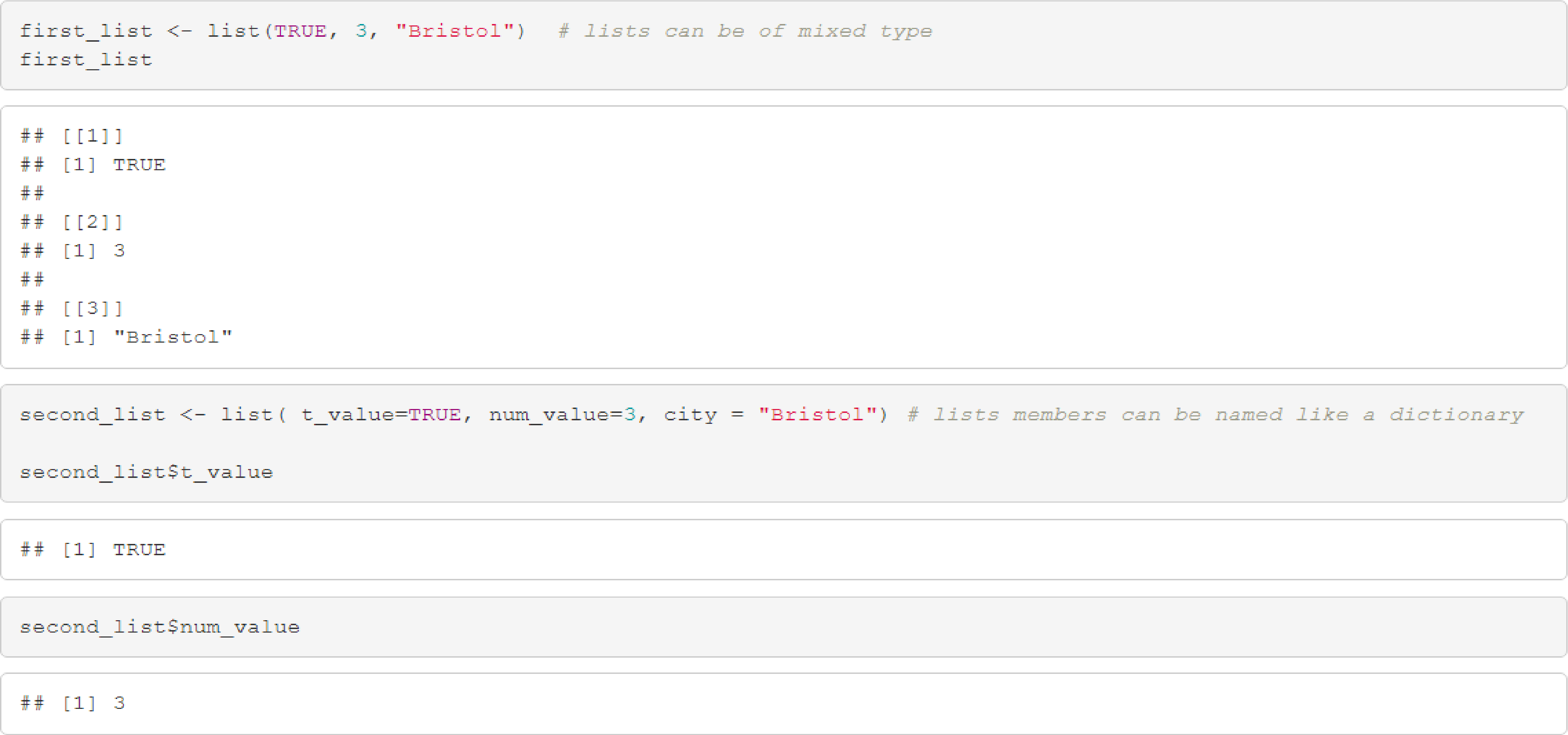
```
a <- c(TRUE, 3, "Bristol") # You can't have a vector of mixed type!
a
```

```
## [1] "TRUE"    "3"       "Bristol"
```

```
mode(a) # You can test the type like this
```

```
## [1] "character"
```

Lists



```
first_list <- list(TRUE, 3, "Bristol")  # lists can be of mixed type
first_list
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] 3
##
## [[3]]
## [1] "Bristol"
```

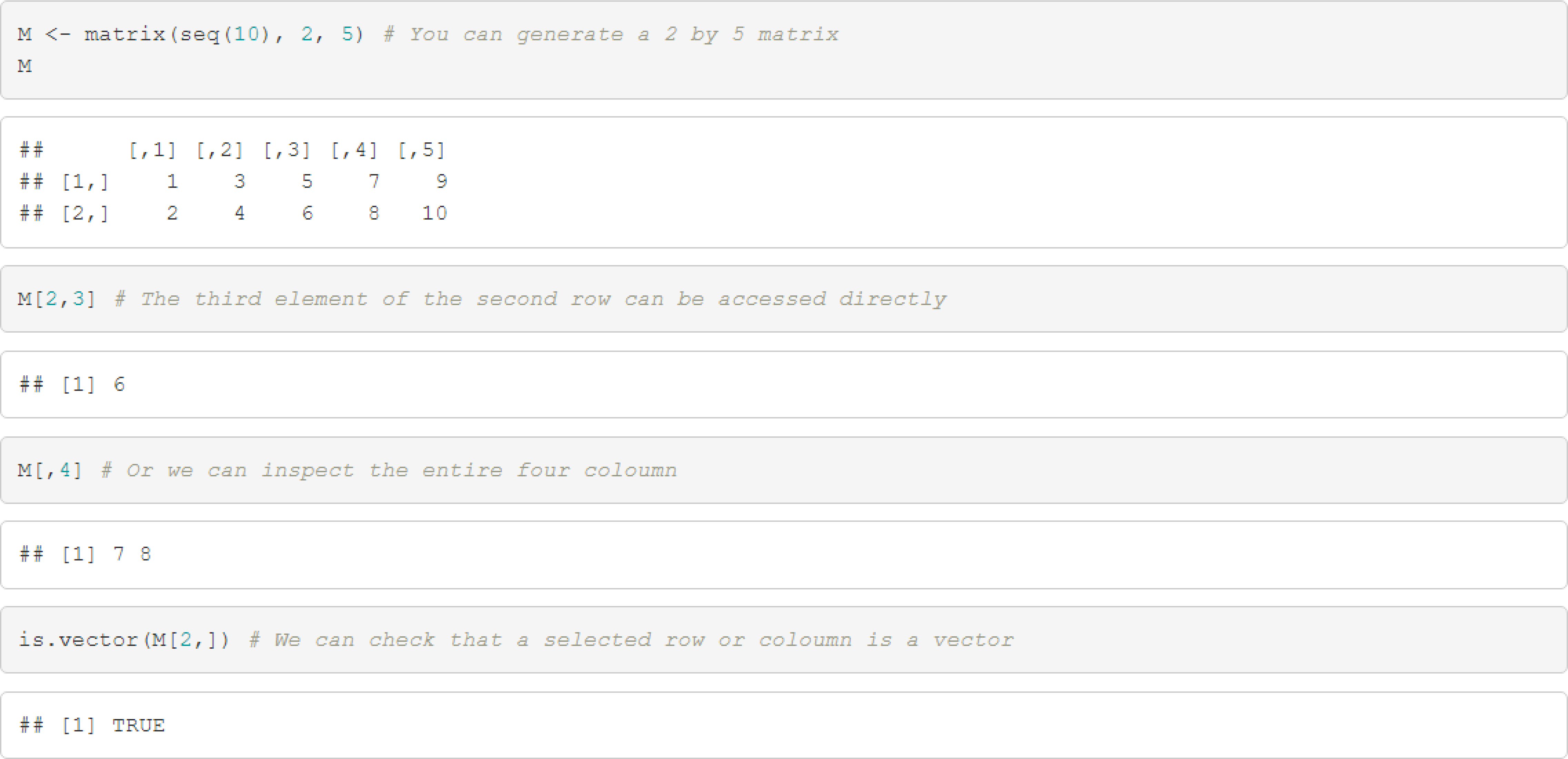
```
second_list <- list( t_value=TRUE, num_value=3, city = "Bristol") # lists members can be named like a dictionary

second_list$t_value
```

```
## [1] TRUE
```

```
second_list$num_value
```

```
## [1] 3
```

Matrices



```
M <- matrix(seq(10), 2, 5) # You can generate a 2 by 5 matrix
M
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
M[2,3] # The third element of the second row can be accessed directly
```

```
## [1] 6
```

```
M[,4] # Or we can inspect the entire four coloumn
```

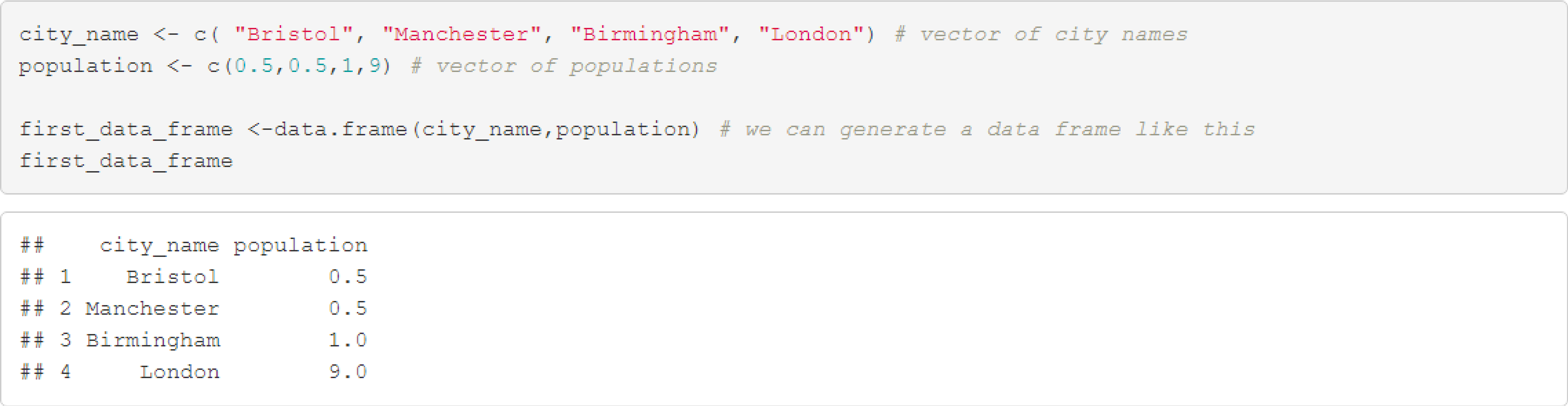
```
## [1] 7 8
```

```
is.vector(M[2,]) # We can check that a selected row or coloumn is a vector
```

```
## [1] TRUE
```

Data frames

- Data frames are powerful objects for representing and manipulating tabular data.

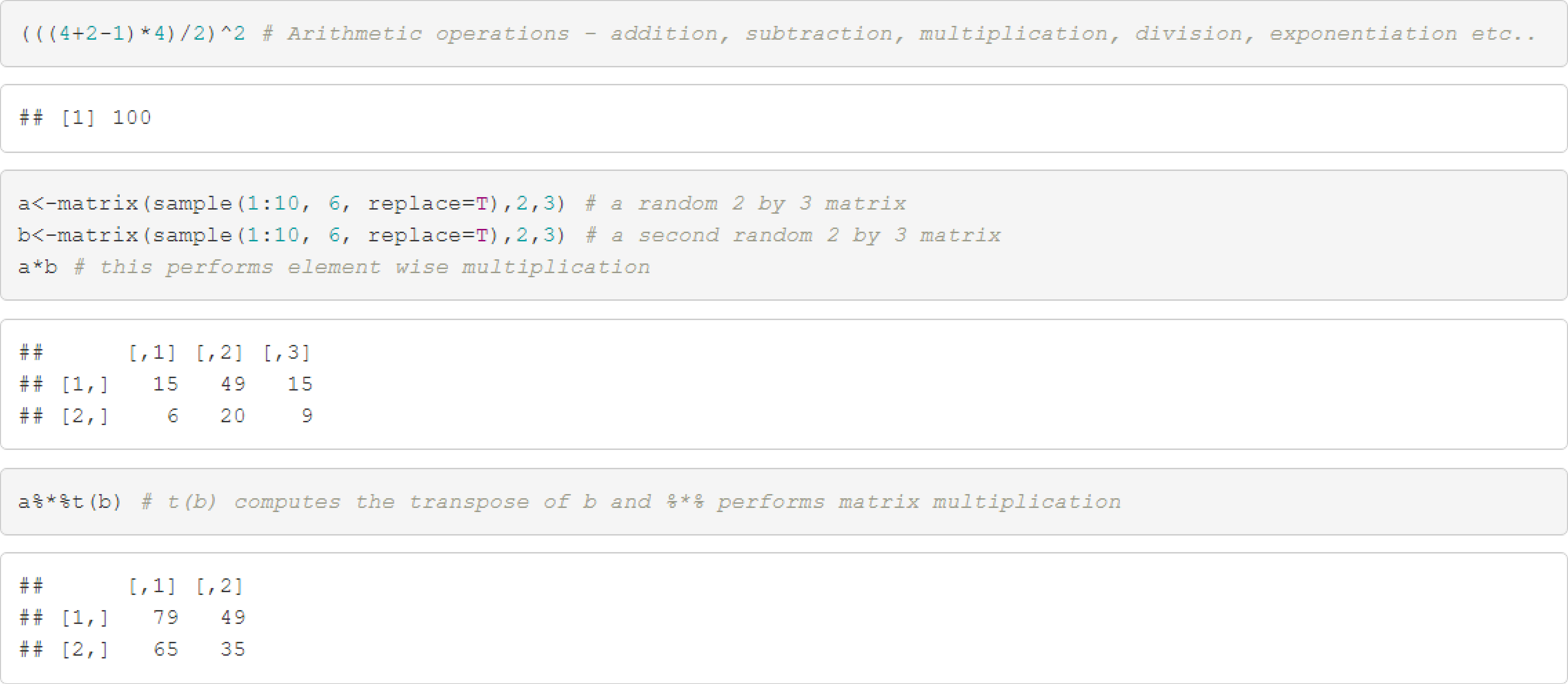


```r
city_name <- c( "Bristol", "Manchester", "Birmingham", "London") # vector of city names
population <- c(0.5,0.5,1,9) # vector of populations

first_data_frame <-data.frame(city_name,population) # we can generate a data frame like this
first_data_frame
```

```
##    city_name population
## 1    Bristol        0.5
## 2 Manchester        0.5
## 3 Birmingham        1.0
## 4     London        9.0
```

- Unlike matrices, columns are named & different columns may be of different type
- However, the cells within a column must be of the same type.

Arithmetic operations



```
(((4+2-1)*4)/2)^2 # Arithmetic operations - addition, subtraction, multiplication, division, exponentiation etc..
```

```
## [1] 100
```

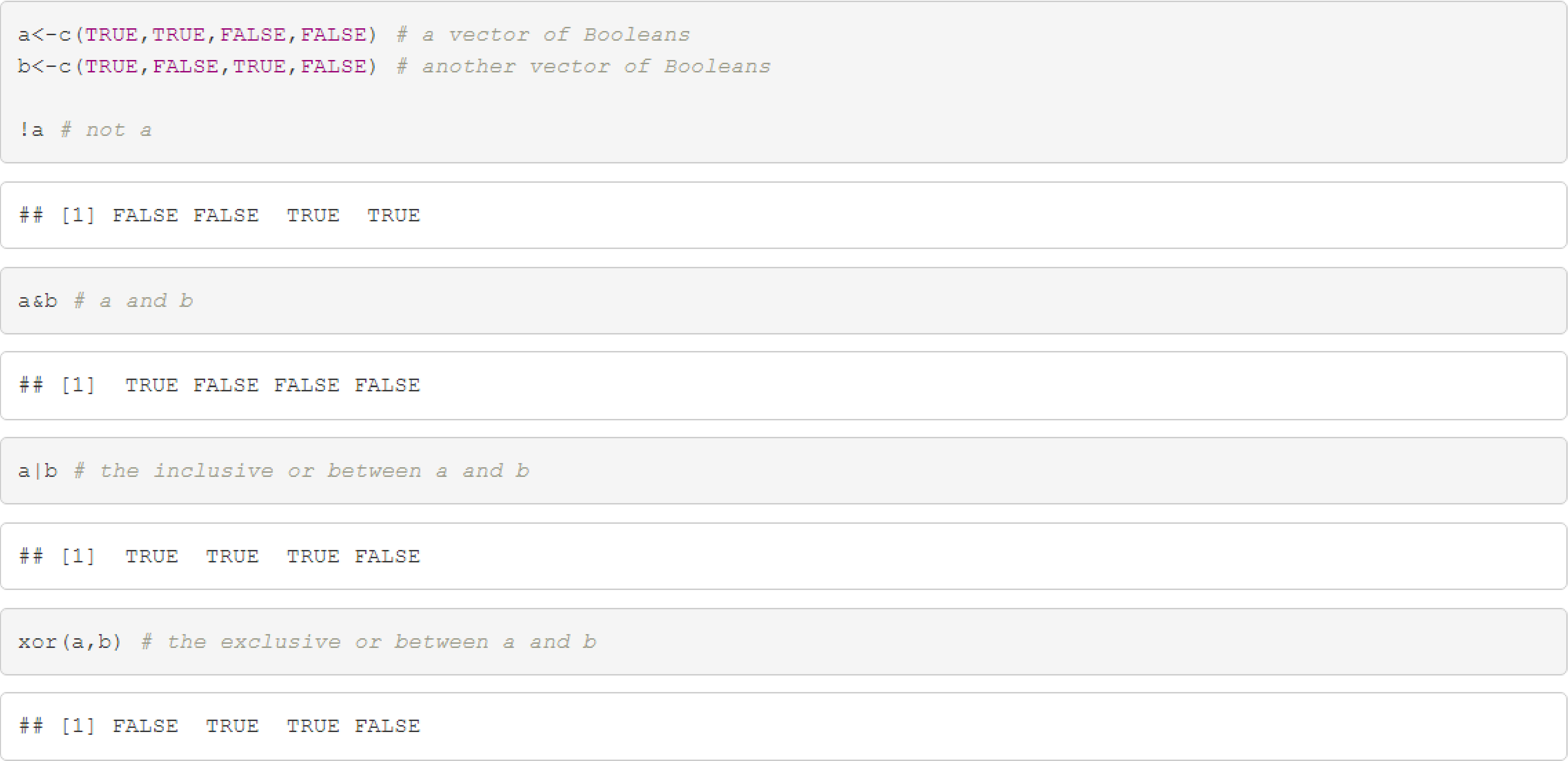
```
a<-matrix(sample(1:10, 6, replace=T),2,3) # a random 2 by 3 matrix
b<-matrix(sample(1:10, 6, replace=T),2,3) # a second random 2 by 3 matrix
a*b # this performs element wise multiplication
```

```
##      [,1] [,2] [,3]
## [1,]   15   49   15
## [2,]    6   20    9
```

```
a%*%t(b) # t(b) computes the transpose of b and %*% performs matrix multiplication
```

```
##      [,1] [,2]
## [1,]   79   49
## [2,]   65   35
```

Boolean operations



```r
a<-c(TRUE,TRUE,FALSE,FALSE) # a vector of Booleans
b<-c(TRUE,FALSE,TRUE,FALSE) # another vector of Booleans

!a # not a
```

```
## [1] FALSE FALSE  TRUE  TRUE
```

```r
a&b # a and b
```

```
## [1]  TRUE FALSE FALSE FALSE
```

```r
a|b # the inclusive or between a and b
```

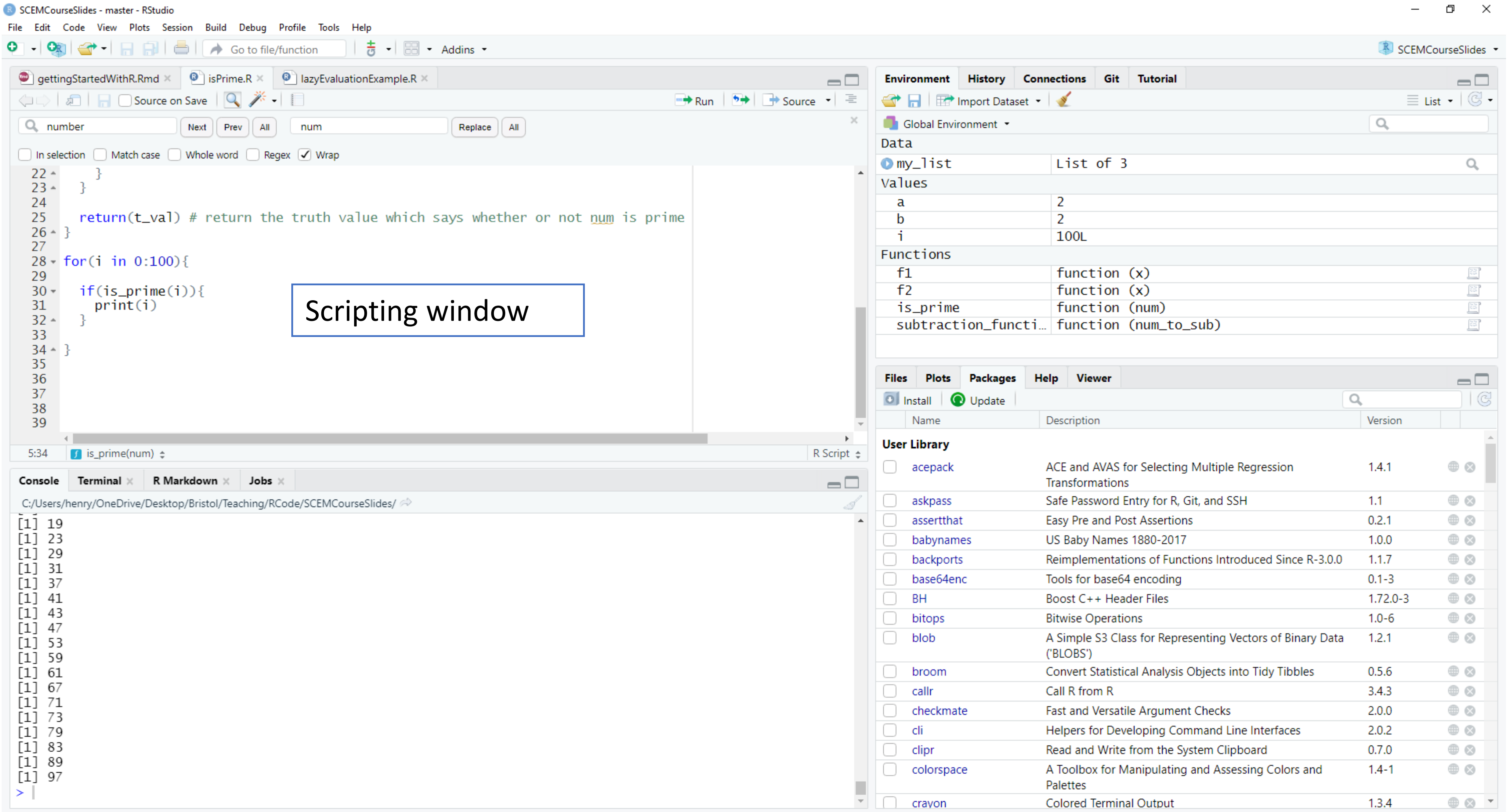
```
## [1]  TRUE  TRUE  TRUE FALSE
```

```r
xor(a,b) # the exclusive or between a and b
```

```
## [1] FALSE  TRUE  TRUE FALSE
```

Now take a break!

Your first R script



number

num

□ In selection □ Match case □ Whole word □ Regex ☑ Wrap

```
    }
  }

  return(t_val) # return the truth value which says whether or not num is prime
}

for(i in 0:100){

  if(is_prime(i)){
    print(i)
  }

}
```

Scripting window

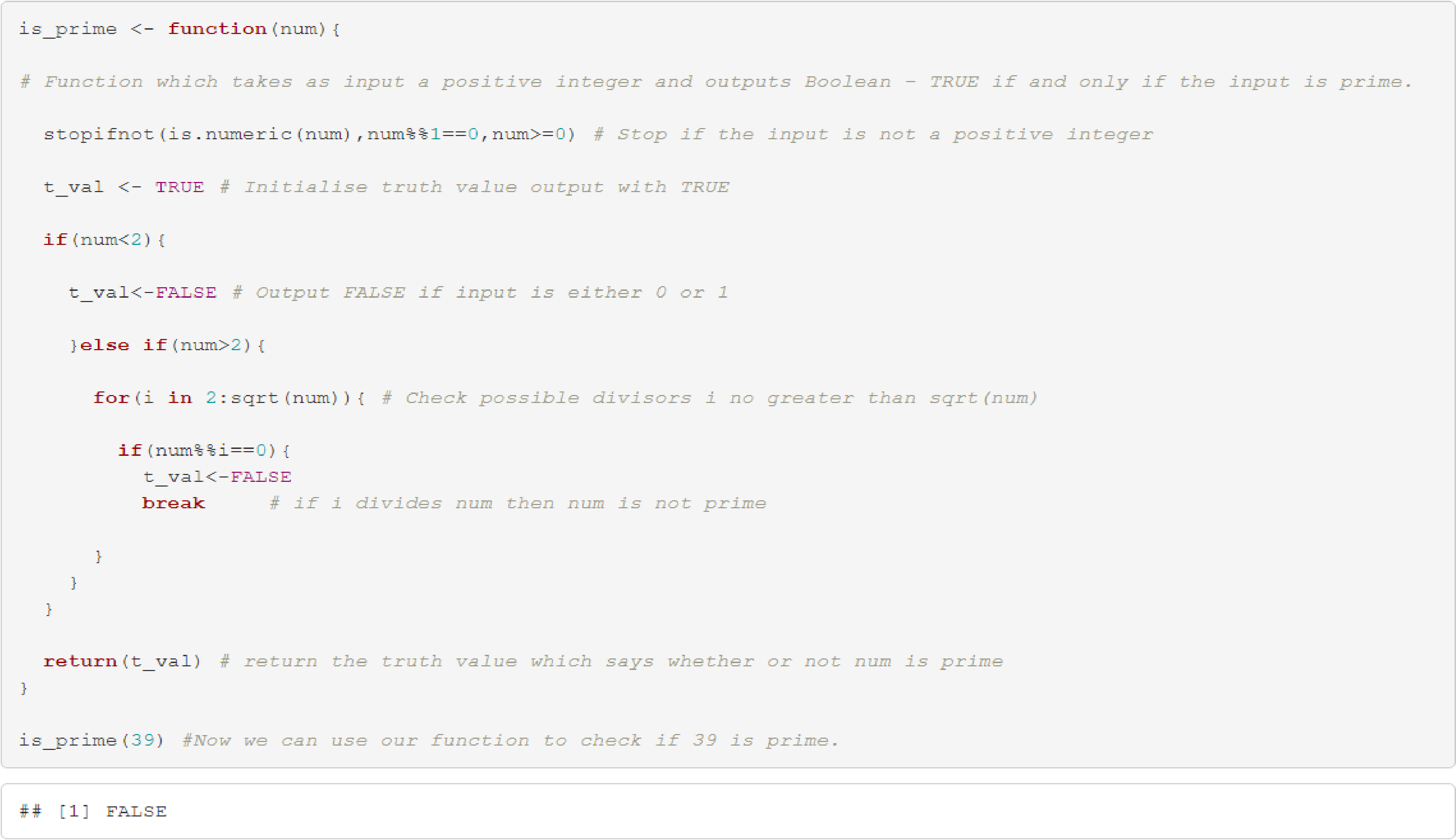
C:/Users/henry/OneDrive/Desktop/Bristol/Teaching/RCode/SCEMCourseSlides/

```
[1] 19
[1] 23
[1] 29
[1] 31
[1] 37
[1] 41
[1] 43
[1] 47
[1] 53
[1] 59
[1] 61
[1] 67
[1] 71
[1] 73
[1] 79
[1] 83
[1] 89
[1] 97
>
```

Data	
my_list	List of 3
Values	
a	2
b	2
i	100L
Functions	
f1	function (x)
f2	function (x)
is_prime	function (num)
subtraction_functi…	function (num_to_sub)

Name	Description	Version
User Library		
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
askpass	Safe Password Entry for R, Git, and SSH	1.1
assertthat	Easy Pre and Post Assertions	0.2.1
babynames	US Baby Names 1880-2017	1.0.0
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.7
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.72.0-3
bitops	Bitwise Operations	1.0-6
blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.1
broom	Convert Statistical Analysis Objects into Tidy Tibbles	0.5.6
callr	Call R from R	3.4.3
checkmate	Fast and Versatile Argument Checks	2.0.0
cli	Helpers for Developing Command Line Interfaces	2.0.2
clipr	Read and Write from the System Clipboard	0.7.0
colorspace	A Toolbox for Manipulating and Assessing Colors and Palettes	1.4-1
crayon	Colored Terminal Output	1.3.4

Functions

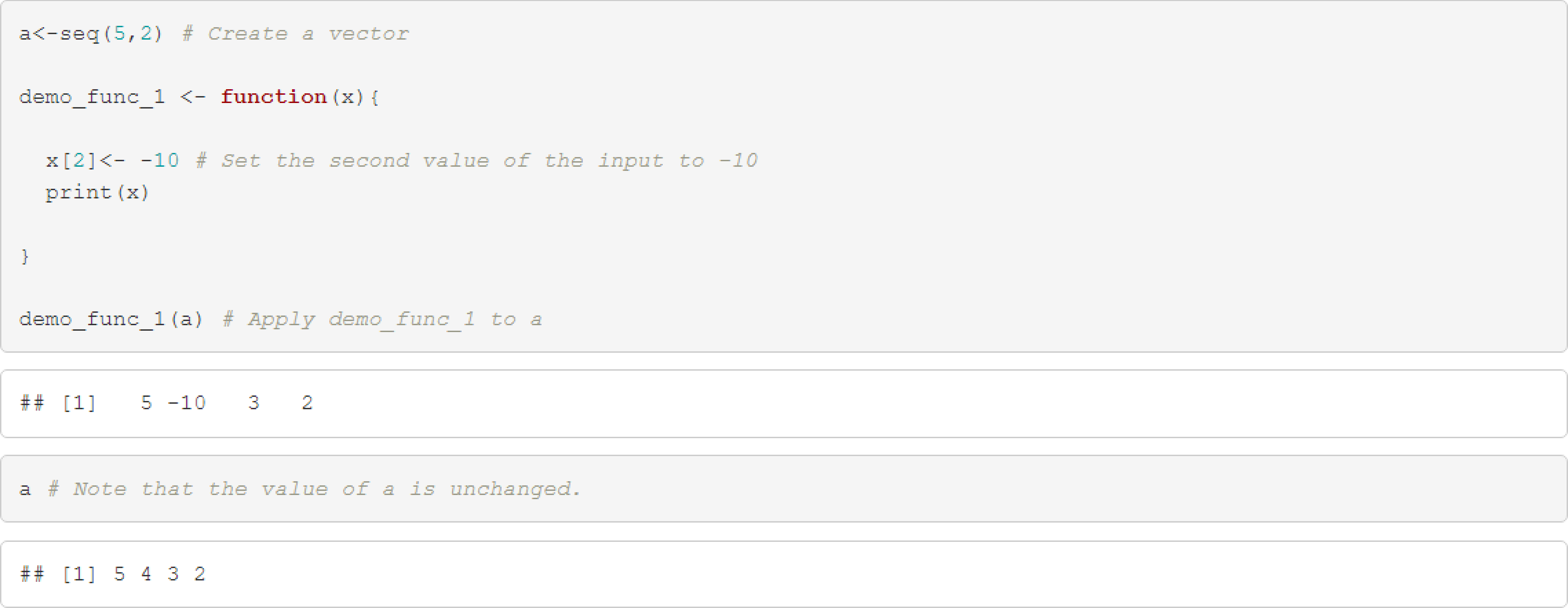


```r
is_prime <- function(num){

# Function which takes as input a positive integer and outputs Boolean - TRUE if and only if the input is prime.

  stopifnot(is.numeric(num),num%%1==0,num>=0)  # Stop if the input is not a positive integer

  t_val <- TRUE # Initialise truth value output with TRUE

  if(num<2){

    t_val<-FALSE # Output FALSE if input is either 0 or 1

    }else if(num>2){

      for(i in 2:sqrt(num)){ # Check possible divisors i no greater than sqrt(num)

        if(num%%i==0){
          t_val<-FALSE
          break     # if i divides num then num is not prime

        }
      }
    }

  return(t_val) # return the truth value which says whether or not num is prime
}

is_prime(39) #Now we can use our function to check if 39 is prime.
```

```
## [1] FALSE
```

Call by value semantics

- In R arguments in functions are passed with call-by-value semantics.



```
a<-seq(5,2) # Create a vector

demo_func_1 <- function(x){

  x[2]<- -10 # Set the second value of the input to -10
  print(x)

}

demo_func_1(a) # Apply demo_func_1 to a
```

```
## [1]   5 -10   3   2
```

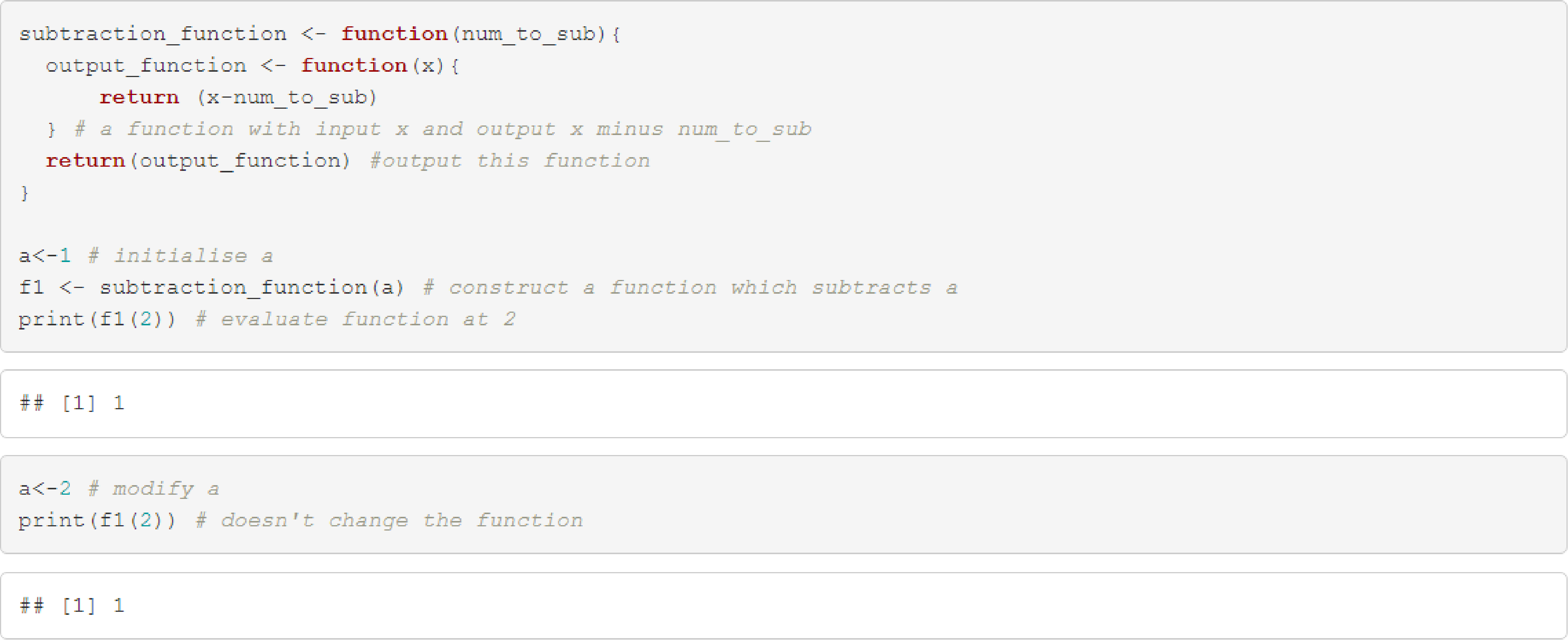
```
a # Note that the value of a is unchanged.
```

```
## [1] 5 4 3 2
```

- This facilitates a functional programming style with limited side effects.

Lazy evaluation

- Lazy evaluation enables efficiency but has some surprising consequences.



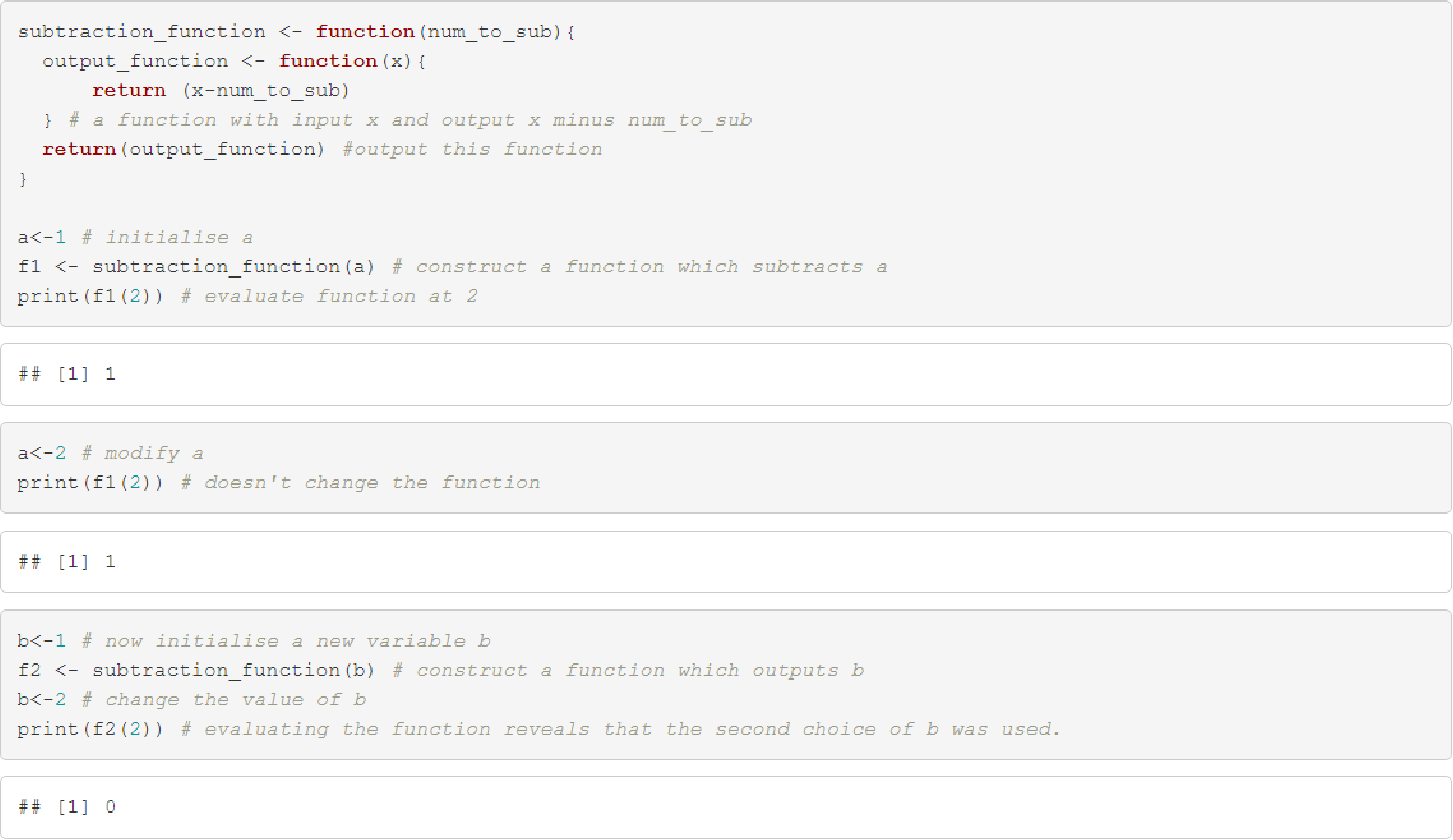
```
subtraction_function <- function(num_to_sub){
  output_function <- function(x){
      return (x-num_to_sub)
  } # a function with input x and output x minus num_to_sub
  return(output_function) #output this function
}

a<-1 # initialise a
f1 <- subtraction_function(a) # construct a function which subtracts a
print(f1(2)) # evaluate function at 2
```

```
## [1] 1
```

```
a<-2 # modify a
print(f1(2)) # doesn't change the function
```

```
## [1] 1
```

Lazy evaluation



```r
subtraction_function <- function(num_to_sub){
  output_function <- function(x){
      return (x-num_to_sub)
  } # a function with input x and output x minus num_to_sub
  return(output_function) #output this function
}

a<-1 # initialise a
f1 <- subtraction_function(a) # construct a function which subtracts a
print(f1(2)) # evaluate function at 2
```

```
## [1] 1
```

```r
a<-2 # modify a
print(f1(2)) # doesn't change the function
```

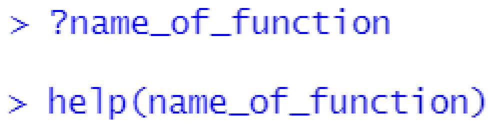
```
## [1] 1
```

```r
b<-1 # now initialise a new variable b
f2 <- subtraction_function(b) # construct a function which outputs b
b<-2 # change the value of b
print(f2(2)) # evaluating the function reveals that the second choice of b was used.
```

```
## [1] 0
```

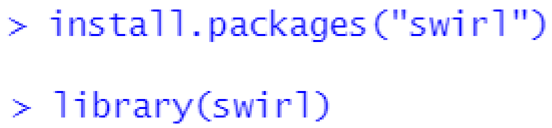
How can we learn more?

- Almost every R function has an associated help function which can be accessed via



```
> ?name_of_function
> help(name_of_function)
```

- A fantastic resource to learn more about R is the Swirl package:



```
> install.packages("swirl")
> library(swirl)
```

- Another great resource is StackOverflow for R:

https://stackoverflow.com/questions/tagged/r

What have we covered?

- We installed R and RStudio and started our first session.
- We learned about the basic objects – vectors, lists, matrices & data frames.
- We considered basic operations – arithmetic and Boolean
- We learnt how to write a simple function with control flow statements
- We saw that R facilitates a functional paradigm with call-by-value semantics
- We briefly looked at lazy evaluation and its surprising consequences
- We discussed a few resources for learning more – the help function, Swirl, stack etc.

That's plenty for today! – now onto the exercises.

University of BRISTOL

Thanks for listening,

… now onto the assignment!

Henry W J Reeve

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.