A Gentle Introduction to R

Jonathan Whiteley

2023-08-13

Prerequisites

- Access to a copy of the R[1] software
  - i.e., a "binary executable"
  - Go to *www.r-project.org* to get a copy, or ask your system administrator.
- No previous experience with R or programming required.

[1] The R logo (R) is © 2016 The R Foundation and used as-is under the terms of the **CC-BY-SA 4.0** license

Section 1

Welcome

Pop Quiz

We will review these *at the end*, so you can see how much you have learned.

- What does 'CRAN' stand for?
- Why is it named '`R`'?
- How can you use `R` *interactively*?
- How do you find out what a function does & how to use it?
- How do you store values to re-use later?
- True or False: Warnings can be ignored, but an Error means I made a mistake.
- True or False: Error messages will tell me how to fix the problem.

Answer in the chat:

What emoji best describes your current mood or state of mind?

Introductions

- Name
- Pronouns
- Job title, role
- *optional*: a hobby or activity you enjoy?

- Have you used `R` before?
- Have you used a programming language before?

Icebreaker activity

What is this?

1–3 word description, for example:

- "This is grey"
- "This looks uncomfortable"

OR caption this image?

On your turn:

1. Previous person's name
2. Their answer to the question
3. Your name
4. Your answer
5. Name of the person to go next

Figure 1: Caption this image.
© John Speirs/Comedywildlifephoto.com

Learning Objectives

- Get familiar with the  *interface*
- Use technical *terms* for R concepts
- Enter *commands*
  - use R interactively: understand input & output
  - use some common *functions*
- Get familiar with 'R objects'
  - store & retrieve values
- Understand *Errors*, *Warnings*, and *Messages*
- How to get Help

Why is it named 'R'?

1. R started as an *open-source* implementation of the S statistical computing language (S-PLUS)[2]
   - S was created at Bell Laboratories in 1976[3]
   - R was based on the S syntax (mostly v3), but works very differently "under the hood".
2. R was created by **R**oss Ihaka and **R**obert Gentleman — aka "R & R"[4] — at the University of Aukland in the early 1990s.

Read more about the history of R on Wikipedia[5]

[2] https://www.r-project.org/about.html
[3] https://en.wikipedia.org/wiki/S_(programming_language)
[4] https://www.r-project.org/contributors.html
[5] https://en.wikipedia.org/wiki/R_(programming_language)#History

Section 2

Interacting with R (Interface)

The R Interface

- 'base R' has a slightly different interface for each **O**perating **S**ystem (OS)
  - GUI = **G**raphical **U**ser **I**nterface
- R can also run inside of a terminal (no GUI) or other software (different GUI).

Integrated Development Environment (IDE)

- An IDE is like an extra interface layer on top of 'base R'
- IDEs often add convenient tools to make writing code easier (e.g., syntax highlighting), and for developing larger projects with multiple files.
- **RStudio** is one of the most popular cross-platform IDEs for R.
  - RStudio is available in open source (free/libre) and commercial[a] editions.

[a] for organizations not able to use software licensed with AGPL

A quick tour of the 'base R GUI'

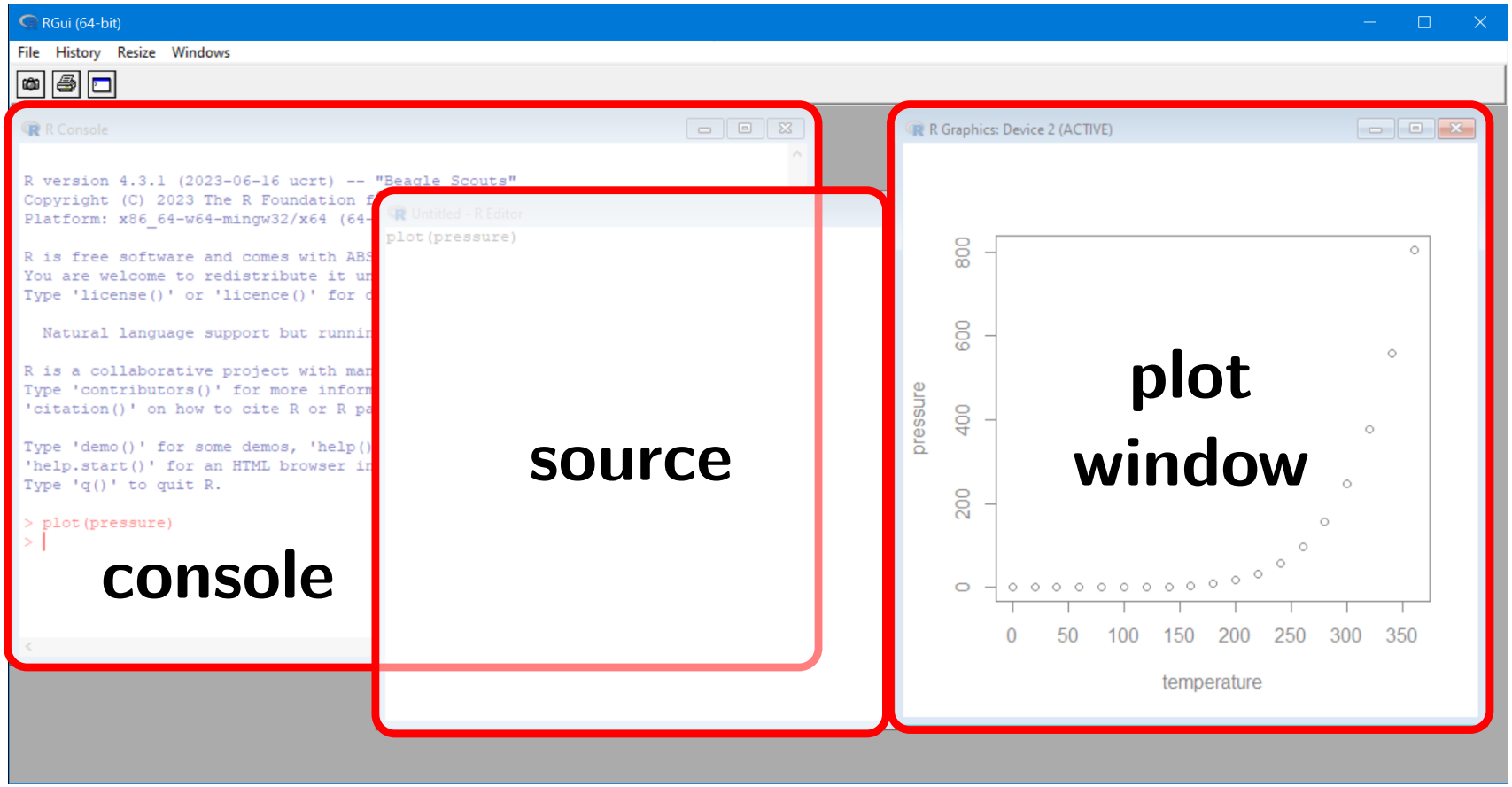


Figure 2: Screenshot of the R GUI in Windows.

A quick tour of RStudio

The RStudio GUI has 4 'panes' that contain 'tabs'.

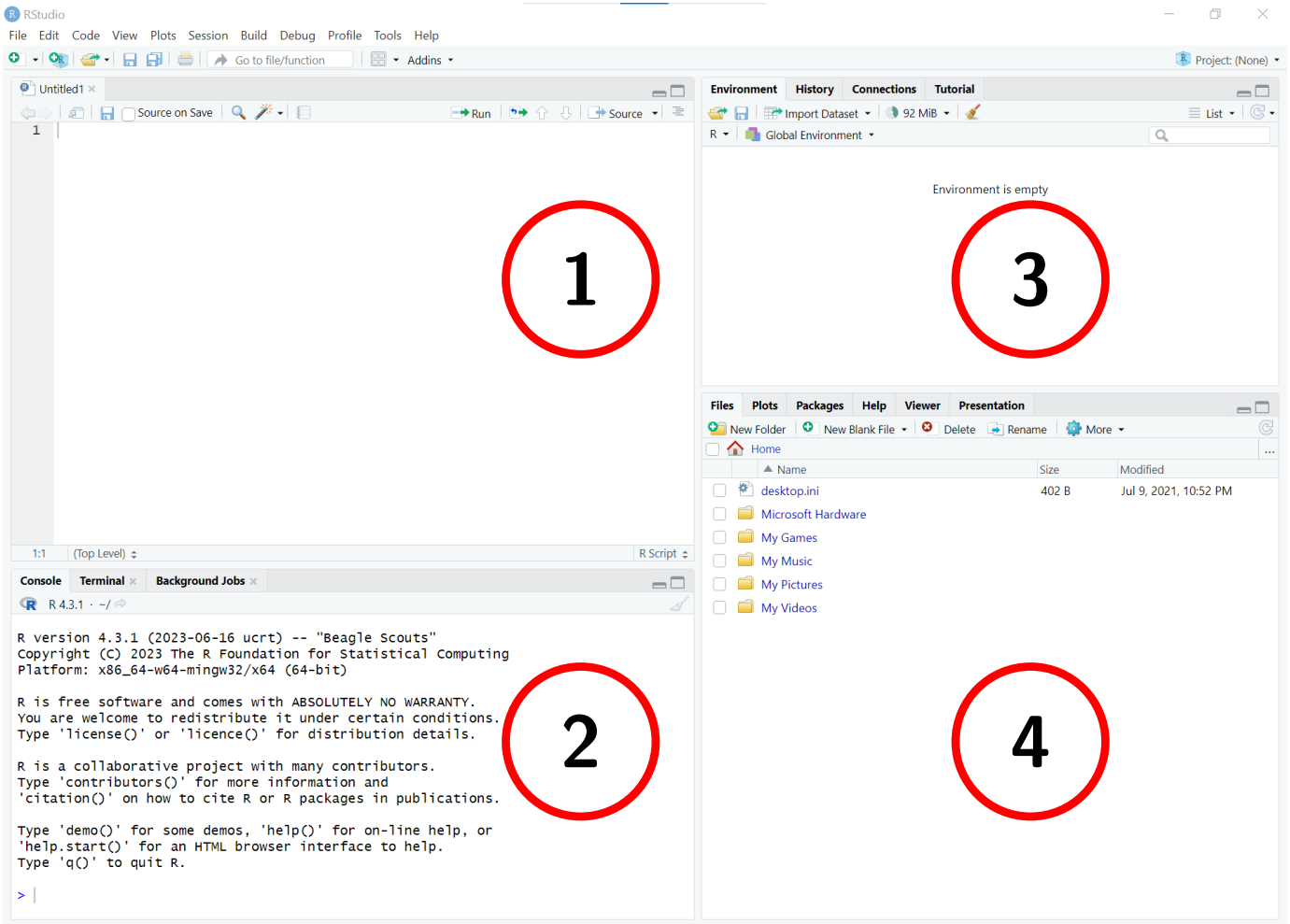


Figure 3: Screenshot of RStudio (default layout).

left:

1. top: **Source**[a]
2. bottom: **Console, Terminal, . . .**

right:

3. top: **Environment, History, . . .**
4. bottom: **Files, Plots, Help, . . .**

[a] empty until you create or open a file

Interacting with R

- Regardless of the GUI, you interact with R primarily using a *command line*
  - aka a command line interface (cli)
  - the command line is usually in the *console*
- "Question-and-Answer Model"
  - You ask R to do something (a *command*), and R tells you the answer (*result*).
- Instructions are given to R using the *R language*.

The R console

The *console* is a window or pane where you will find:

- The *command line*
  - where you will enter commands for `R` to run
- Results of commands and other output
- `Messages`, ***Warnings***, and **Errors**

The R command-line

- The command *prompt* normally looks like this:

```
>
```

(the colour varies depending on the interface)

  - This is R's way of saying "I am ready to accept new commands".
  - Type a new command on the line after this prompt (i.e., *input*).

- **Press `return`/`enter` to *run* the current *command***

- If you can still edit the command next to the prompt, then it has not been submitted to R to execute (it is still waiting for input).

- If the last prompt is not empty (i.e., there is text beside it) *and* you cannot edit what is beside the prompt, it means R is still running the last command and is not ready to accept a new command yet.

The R command-line (continued)

- If the prompt looks like this:

```
+
```

  it means the last command was *incomplete* and R is waiting for more input.
  R will not do anything until the command is completed or cancelled.

  - This usually means you forgot a closing
    quote ", *parenthesis* (, *bracket* [, or *brace* {

- You can *cancel* the current command at any time by pressing escape (esc)

Section 3

Warming up: some early commands

Input & Output

In this presentation,

- *commands* that can be entered in the *command-line* look like this:

```
Input (commands)
```

  - You can try these yourself!

- Expected output (results) look like this:

```
Output (results)
```

R offers suggestions

Read the opening message carefully.

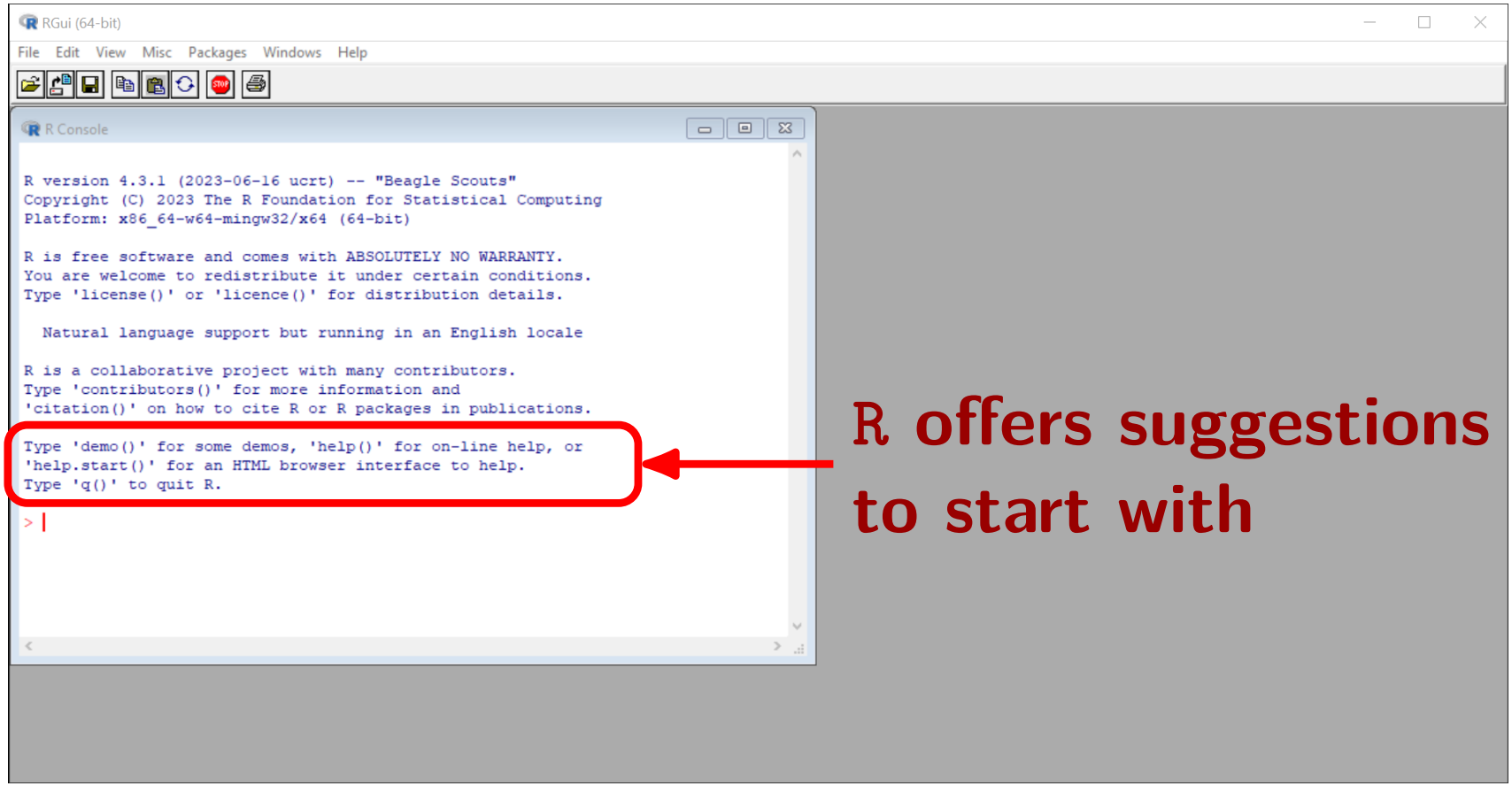


Figure 4: R offers suggestions of commands to Type in the console when it starts.

R is a show-off

```
demo(graphics)
```

- some plots and graphs that can be made with R

```
demo(image)
```

- image-like graphics and maps that can be produced with R

```
demo(lm.glm)
```

- a demonstration of linear modelling & GLMs

```
demo()
```

- a list of available demos

```
help.start()
```

← A great place to start,
especially if you are comfortable reading documentation for a programming language.
More on this later.

Note

R will not only show the output, but also *the code used to produce it*.

R is a show-off (alt)

```
demo(graphics)
demo(image)
demo(lm.glm)
demo()

help.start()
```

- some plots and graphs that can be made with R
- image-like graphics and maps that can be made with R
- a demonstration of linear modelling & GLMs
- a list of available demos

A great place to start, especially if you are comfortable reading documentation for a programming language. More on this later.

Note

R will not only show the output, but also *the code used to produce it*.

R is a calculator

```
1 + 1
```

```
[1] 2
```

```
2 * 2
```

```
[1] 4
```

```
2 ^ 3
```

```
[1] 8
```

```
10 - 1
```

```
[1] 9
```

```
8 / 2
```

```
[1] 4
```

```
sqrt(9)
```

```
[1] 3
```

- These are *expressions*
- *Expressions* are *evaluated*, and the *value* (result) is *returned* (sometimes *invisibly*)

command-line tips

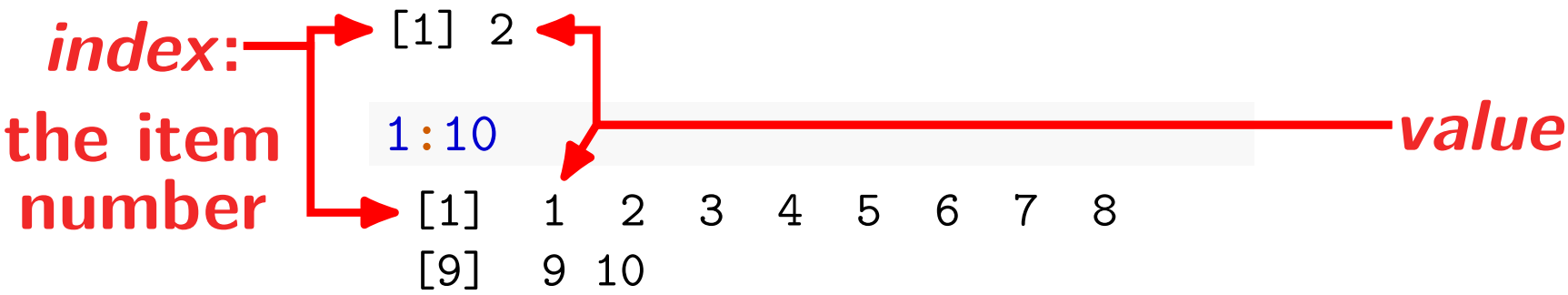
- With the cursor next to the empty prompt (>),
  use the up & down arrow keys (↑↓) to re-produce previous commands.
- This lets you "scroll through your *command history*".
- Press up (↑) once, and you get the last command you entered
  without having to copy & paste.

Section 4

Simple R objects

Vectors

- The most basic kind of *object* in R is a *vector*
- Think of a vector as a list of related values (data), which are *all the same type*
- A single value is an "*atomic vector*" (a vector with a length of 1)



Using vectors

- Vectors can be used in calculations
- Operations are applied to each item (*element-wise*)

```
sum( c(1, 2, 3, 4, 5) )
1:10 + 2
1:5 * 5:1
```

- Vectors can be used to plot data in a graph

```
plot( rnorm(1000) )
hist( rnorm(1000) )
```

Some data *types* (of *atomic vectors*)

numeric

- Includes *integers*, *real* (decimal / *double*), and *complex* numbers.
- `1.23`

character (*string*)

- in single `'` or double `"` quotes.
- `'hello world'`
- `"1.23"`

logical

- `TRUE` or `FALSE`

```
class(1.23)
class('hello')
class("1.23")
class(FALSE)

typeof(1.23)
typeof(1:10)

as.character(c(1,2,NA,4))
```

`as.*()`: converting from one type to another = *coercion*

Section 5

Storing & retrieving values

Symbolic *variables*

- You can store values (*objects*) in symbolic variables (*names*) using an *assignment operator*:

`<-`	assign the *value* on the **right** to the *name* on the **left**

- Names can include:

letters	`a-z A-Z`
numbers	`0-9`
periods	`.`
underscores	`_`

```
A <- 10
B <- 10 * 10
A_log <- log(A)
B.seq <- 1:B

assign('x', 3)
```

- Names *should begin with a* ***letter***.

Retrieve values

When a variable *name* is evaluated, it returns the stored *value*.

```
A
```

```
[1] 10
```

```
B
```

```
[1] 100
```

```
A_log
```

```
[1] 2.303
```

```
x
```

```
[1] 3
```

```
B.seq
```

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

Built-in variables

Some words and letters already have values in R and should **never be used as variable names**.

```
pi
```

```
[1] 3.142
```

```
version
```

```
... information about
this version of R ...
```

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
LETTERS
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

Reserved words

Some words and letters already have special meaning in the R language (*keywords*) and should **never be used as variable names**.

`NA`	"Not Available"	placeholder for unknown or missing values
`NaN`	"Not a Number"	placeholder for *undefined* numeric values
`NULL`	*a special object*	placeholder for missing *objects*
`Inf`	Infiniti	
`TRUE`	Logical value	
`FALSE`	Logical value	
`T`	short for `TRUE`	
`F`	short for `FALSE`	
`c,q,t,C,D,I`	`R` functions	
`diff, df, pt`	`R` functions	

R is case-sensitive

`R.version`	a variable	`pi`
`R.Version()`	a *function*	*PI*
`letters`	a-z	`NA`
`LETTERS`	A-Z	*na*

Use variables in calculations

```
A +5
```

```
[1] 15
```

```
B/A
```

```
[1] 10
```

```
Weight <- c(60 , 72 , 57 , 90 , 95 , 72 )
Height <- c(1.7, 1.8, 1.6, 1.9, 1.7, 1.9)
BMI <- Weight / Height^2
BMI
```

```
[1] 20.76 22.22 22.27 24.93 32.87 19.94
```

```
plot(Height, Weight)
```

Housekeeping

`ls()`	List all variables you have created
`rm(x)`	Remove the variable 'x' from memory
`rm(list=ls())`	Remove *all variables* from memory (clear memory)

```
pi
pi <- "pie"
pi
rm(pi)
pi
```

Section 6

Operators

Operators

Operators are special symbols that go between two values, to perform an *operation* on both values (the *operands*) and return the *result*.

- For example: `2 * 3` is a way of saying "*multiply* 2 and 3 together"
- Operations are evaluated one pair at a time, according to precedence (*order of operations*).

Arithmetic Operators

The usual math symbols:
`+`, `-`, `*`, `/`, `^`, etc.

Assignment Operators

Assign values to symbolic variables:
`<-`, `->`, `=`, etc.

Comparison (*Relational*) Operators

For comparing two values:
`==`, `!=`, `>`, `<`, etc.

Boolean Operators

Combining logical values (`TRUE`, `FALSE`): `!`, `&`, `|`, etc.

Comparisons

Comparison of 2 values results in *logical values*: `TRUE` or `FALSE`

`==`	"equal" — Note the **two** equals signs. Not to be confused with a single equals sign (used to *assign* values).
`!=`	"not equal"
`>`	"greater than"
`<`	"less than"
`>=`	"greater than or equal to"
`<=`	"less than or equal to"

Comparisons: examples

```
1 == 2
```

```
[1] FALSE
```

```
1 < 2
```

```
[1] TRUE
```

```
1 <= 2
```

```
[1] TRUE
```

```
1 != "foo"
```

```
[1] TRUE
```

```
1 < "a"
```

```
[1] TRUE
```

```
0 == FALSE
```

```
[1] TRUE
```

Comparing decimals ('floating point' arithmetic)

Computers can't represent *all* values accurately, and there is often some rounding that occurs (even at 50+ decimal places). As a result, 'floating point' values may not be *reliably equal*. [6] [7]

This is a common source of confusion, but it is a fact of how computers handle floating point arithmetic, and not specific to R.

```
a <- sqrt(2)
a * a == 2  # should be TRUE
```

```
[1] FALSE
```

```
a * a - 2
```

```
[1] 4.441e-16
```

Two common solutions: 1. `round()` decimal values when comparing them 2. use a function with a tolerance for small differences, such as `all.equal()`

```
round(a * a, 8) == 2
```

```
[1] TRUE
```

```
all.equal(a * a, 2)
```

```
[1] TRUE
```

[6] R FAQ: "Why doesn't R think these numbers are equal?"

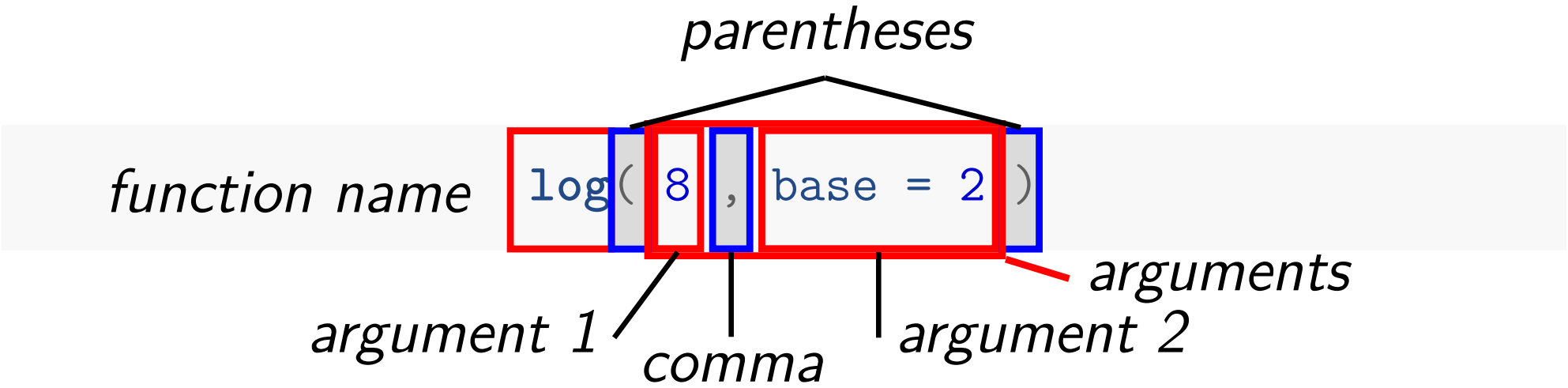
[7] See Stackoverflow: "Why are these numbers not equal?" for other solutions

Section 7

Functions

Functions

- *Functions* are special commands that can do more than simple operators[8].
- They are the main instructions you give to R.
- To use (or *call*) a function, the command must be structured properly, following the "grammar rules" of the R language (*syntax*).



[8] technically, operators are special functions with exactly 1 (*unary*) or 2 (*binary*) *arguments*. See section 3.1.4 "Operators" in the R Language Definition.

Function arguments

- *arguments* are the values passed to a function when it is *called*
  - these are values the function needs to do its thing
  - some change *how* the function operates (these are usually optional)
- arguments can be *passed by order* or *passed by name*
  - *passed by order* means the arguments are specified in the correct order
  - *passed by name* means the arguments can be in any order, but must be declared by name.
    e.g.: `argument = value`

Section 8

Errors, Warnings, and Messages

Errors

- When R receives a command it does not understand, or cannot execute, it outputs an ***error*** to the *console*.
  - This is in the form of a message that begins with the word "`Error`".
- A command that produces an *error* is **not** executed.
  - neither are any commands after the error.

```
Fail <- 1 + "2"
```

```
Error in 1 + "2" : non-numeric argument to binary operator
```

```
Fail
```

```
Error in eval(expr, envir, enclos) : object 'Fail' not found
```

- When an error occurs, `R` **stops running** commands and returns to the command-line.
  - Your *session* is still active: `R` didn't quit, and you can enter more commands.

Warnings

- Some commands still work, but did not run exactly as R (or the developers) think is "ideal", and may produce a ***warning*** instead.
  - This is in the form of a message that begins with the word "***Warning***".
- These do not interrupt what R is doing: it will keep running, but tell you that there were warnings.
  - *It is up to you to review the warnings and decide if they are important.*
  - Use the `warnings()` command to review them.

```
oops <- log(-1)
```

```
Warning in log(-1): NaNs produced
```

Messages, Warnings, and Errors

- ***Messages*** are for information, and a sign that things are working fine (at least, according to the programmers who created the function).
  - Think of messages as a green traffic light: you are safe to continue.
- ***Warnings*** indicate something unusual happened, but `R` is able to continue. You'll have to assess if it's worth worrying about.
  - Think of warnings as a yellow traffic light: you can go, but be careful and pay attention, in case there is a problem.
- ***Errors*** indicate something is wrong, and `R` had to stop. You'll have to figure out what caused the error, fix it, and try again.
  - Think of errors as a red traffic light: stop — something is wrong!

Section 9

Help & documentation

Section 10

Installing packages

Section 11

Saving code (files)

Saving code (files)

Section 12

Backmatter

Quiz Review

References & More Information

```
help.start()
```

Accessible from the screen above (offline):

- An Introduction to R
- The R Language Definition

Online:

- RStudio Education (education.rstudio.com)
  - tutorials, workshop materials, and other resources.
-  Manuals (https://cran.r-project.org/manuals.html)
-  Contributed Documentation
  - e.g., http://cran.r-project.org/doc/contrib/usingR.pdf
- Internet search
  - Stack Overflow (stackoverflow.com)
  - Cookbook for R (www.cookbook-r.com)