2018-09-05-dundee Software Carpentry: `R` lesson speaker notes

These notes are for the tutor(s) on the first morning session of the Software Carpentry course held on 5-7th September 2018 at the University of Dundee, teaching the refresher in `R`.

Learning objectives

- Introduction/refresher for `RStudio`
  - understand what `RStudio` is
  - know the main windows of `RStudio` and what functions they provide
- Introduction/refresher for `RStudio` and `git`/`GitHub` project setup
  - create a project in `RStudio`
  - use good practice for project layout in `RStudio`
  - place a project under `git` version control with `RStudio`
- Refresher for flow control in `R`
  - understand and use `if()`...`else()` statements
  - understand and use `for()` loops
  - understand and use `while()` loops
- Refresher for functions in `R`
  - understand the composition of an `R` function
  - how to call functions

  - how to write functions
  - understand when to write functions for good code structure
- Introduction to `RMarkdown` and `knitr`
  - understand the purpose of literate programming
  - understand what a Markdown document is
  - understand and be able to use `RMarkdown` syntax
- Good programming practice
  - good choices for variable names
  - understand the importance of good documentation
  - when and how to write comments in code

## Prerequisites

We assume that the learners have prior exposure to many concepts:

- `R`
- variables and variable assignment
- `R` data types and data structures, especially `data.frame`s
- using `R` packages
- `R` base graphics and `ggplot2`

Things to remember

Clearing the console in `R`

- remove all variables

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

Get a 'clean' console

```
CTRL + L
```

SLIDES

TITLE: Programming in `R`

ETHERPAD

- **DEMONSTRATE LINK AND PAGE**

- Please use the course etherpad to

  - make notes
  - ask questions (someone will be looking at the page)
  - share your knowledge with the rest of the class
  - relive the class afterwards

LEARNING OBJECTIVES

- We're being **QUITE AMBITIOUS**, but we've a lot of time this morning, so should be OK
- We're covering some **FUNDAMENTALS OF `RSTUDIO`**
  - **CREATING** projects and **PUTTING UNDER VERSION CONTROL**
- We're covering some **FUNDAMENTALS OF PROGRAMMING** in `R`, but principles that are **APPLICABLE TO ANY LANGUAGE**
- We're learning some **BEST PRACTICES FOR WRITING AND ORGANISING CODE**
- Much of the morning session is **INTENDED AS A REFRESHER**
- We'll be **ASSUMING YOU ALREADY USE `R`** so are familiar with some aspects:
  - `R` syntax
  - data types and data structures (e.g. `data.frame`s)
  - variables, and variable assignment
  - `R` packages
  - `R` base graphics and `ggplot2`
- **IF ANYTHING IS NEW OR UNCLEAR, PLEASE ASK STRAIGHT AWAY**

SECTION 01: `RSTUDIO`

LEARNING OBJECTIVES

- We're going to cover the **BASIC ELEMENTS OF AN `RSTUDIO` SESSION**
- How `RStudio` **HELPS WITH LIVE ANALYSES**
- How `RStudio` **HELPS WITH WRITING CODE FOR REPRODUCIBLE ANALYSIS**

WHAT IS `RSTUDIO`?

- `RStudio` is an **INTEGRATED DEVELOPMENT ENVIRONMENT** (IDE)
  - available on **ALL MAJOR OPERATING SYSTEMS**
  - available **AS A WEBSERVER**
- On the left is a Mac screenshot, Windows on the right
- `RStudio` provides **PANES** so you can:
  - write **LIVE CODE** (console pane)
  - **VISUALISE AND QUERY DATA LIVE** (graphics and environment pane)
  - write **SCRIPTS AND DOCUMENTS FOR REUSE** (editor pane)

  - **MANAGE PROJECTS AND FILES** (file/`git` panes)

`RSTUDIO` OVERVIEW - **INTERACTIVE DEMO**

- **REMIND PEOPLE THEY CAN USE RED/GREEN STICKIES AT ANY TIME**
  - **(INTRODUCE RED/GREEN STICKIES IF NECESSARY)**
- **ASK PEOPLE TO START `RSTUDIO`**
  - There will be problems. Deal with them, now. It's OK if a couple of people are getting help when you start.

🟥 Red sticky for a question or issue 🟩 Green sticky if complete

- **DESCRIBE THE STARTING VIEW OF `RSTUDIO`**
- You should see **THREE PANELS**
  - Interactive `R` **CONSOLE**: **type here and get instant feedback**
  - **ENVIRONMENT/HISTORY** window
  - Files/Plots/Packages/Help/Viewer: **interacting with files on the computer, and viewing help and some output**
- **REMEMBER THE WINDOWS ARE MOBILE AND PEOPLE COULD HAVE THEM IN ANY CONFIGURATION - THE EXACT ARRANGEMENT IS UNIMPORTANT**
- We're going to use `R` in the interactive console to get used to some of the features of the language, and `RStudio`.
  - **THE RIGHT ANGLED BRACKET IS A PROMPT**: `R` expects input
  - Type calculations, then press `return`
- **DEMO CODE: ASK PEOPLE TO TYPE ALONG**

```
> 1 + 100
[1] 101
> 30 / 3
[1] 10
```

- **RESULT IS INDICATED WITH A NUMBER** `[1]` this indicates the line with output in it
- If you type an **INCOMPLETE COMMAND**, `R` will wait for you to complete it with the prompt `+`
- **DEMO CODE**

```
> 1 +
+
```

- The **PROMPT CHANGES TO `+` WHEN `R` EXPECTS MORE INPUT**
- You can either complete the line, or use `Esc` (`Ctrl-C`) to exit

```
> 1 +
+ 6
[1] 7
> 1 +
+

>
```

- `R` obeys the usual **PRECEDENCE OPERATIONS** (`(`, `**`/`^`, `/`, `*`, `+`, `-`)
- **DEMO CODE**
  - **NOTE SPACES AROUND OPERATORS**

```
> 3 + 5 * 2
[1] 13
> (3 + 5) * 2
[1] 16
> 3 + 5 * 2 ^ 2
[1] 23
> 3 + 5 * (2 ^ 2)
[1] 23
```

- **ARROW KEYS** recover old commands

- The **`HISTORY` TAB** shows all commands used
- `R` will report in **SCIENTIFIC NOTATION**
  - **CHECK THAT EVERYONE KNOWS WHAT SCIENTIFIC NOTATION IS**

🟥 Red sticky for a question or issue 🟩 Green sticky if complete

```
> 2 / 1000
[1] 0.002
> 2 / 10000
[1] 2e-04
> 5e3
[1] 5000
```

BUILT-IN FUNCTIONS

- R has many **STANDARD MATHEMATICAL FUNCTIONS**
- **FUNCTION SYNTAX**
  - type the function name
  - open parentheses
  - type input value
  - close parentheses
  - press return
- **DEMO CODE** - ask for example functions

```
> sin(1)
[1] 0.841471
> log(1)
[1] 0
> log10(10)
[1] 1
> log(10)
[1] 2.302585
```

GETTING HELP FOR BUILT-IN FUNCTIONS

- How do we learn more about a function, or the difference between `log()` and `log10()`?
  - **USE R BUILT-IN HELP**
- **DEMO CODE**

```
> ?log
> help(sin)
```

- This brings up help in the **HELP WINDOW**
  - Scroll to the bottom of the page to find **EXAMPLE CODE**
- You can also use the **SEARCH BOX** at the top of the help window (try `reduce`)

```
> ??log
> args(log)
function (x, base = exp(1))
NULL
> args(log10)
function (x)
NULL
```

- If you're not sure about spelling, the editor has **AUTOCOMPLETION** which will suggest all possible endings for something you type (try `chartr`)

- **USE TAB TO SEE AUTOCOMPLETIONS FOR VARIABLES**

```
> myvar = 10
> myv[TAB]
```

### NUMERICAL COMPARISONS

- We can do **COMPARISONS** in `R`
  - Comparisons return `TRUE` or `FALSE`.
- **DEMO CODE**

```
> 1 == 1
[1] TRUE
> 1 != 2
[1] TRUE
> 1 < 2
[1] TRUE
> 1 <= 1
[1] TRUE
> 1 > 0
[1] TRUE
> 1 >= -9
[1] TRUE
```

- **NOTE:** when comparing numbers, it's better to use `all.equal()` (*machine numeric tolerance*) **ASK IF THERE'S ANYONE FROM MATHS/PHYSICS/COMPUTER SCIENCE**

```
> pi - 1e-8 == pi
[1] FALSE
> all.equal(pi, pi - 1e-8)
[1] TRUE
> all.equal(1.0, 1.0)
[1] TRUE
> all.equal(1.0, 1.1)
[1] "Mean relative difference: 0.1"
> ?all.equal
> all.equal(pi, pi - 1e-8)
[1] TRUE
> all.equal(pi, pi - 1e-8, 1e-16)
[1] "Mean relative difference: 3.183099e-09"
> all.equal(pi, pi - 1e-32)
[1] TRUE
> all.equal(pi, pi - 1e-32, 1e-16)
[1] TRUE
# The precision is set as the square root calculation below - this may
differ from machine to machine
> .Machine$double.eps
```

```
[1] 2.220446e-16
> sqrt(.Machine$double.eps)
[1] 1.490116e-08
```

- **THE ORDER/CONSTRUCTION OF MATHEMATICAL OPERATIONS CAN MATTER**
    - Write somewhere if possible: $a = ¥log(0.01^{200})$, $b = 200 ¥times ¥log(0.01)$
    - These two mathematical expressions are exactly equal: $a = b$
    - But computers are not mathematicians, they're machines. Numbers are susceptible to this *rounding error*, so what happens is this:

```
> log(0.01 ^ 200)
[1] -Inf
> 200 * log(0.01)
[1] -921.034
```

- **COMPUTERS DO WHAT YOU TELL THEM, NOT NECESSARILY WHAT YOU WANT**

WORKING IN RSTUDIO

- `RStudio` offers **SEVERAL WAYS TO WRITE CODE**
    - We'll not see all of them today
    - You've seen **DIRECT INTERACTION IN THE CONSOLE** (entering variables)
    - `RStudio` also has an editor for writing scripts, notebooks, markdown documents, and Shiny applications (**EXPLAIN BRIEFLY**)
    - It can also be used to write plain text

- **INTERACTIVE DEMO OF `R` SCRIPT**

- Click on `File` -> `New File` -> `Text File`. **NOTE THAT THE EDITOR WINDOW OPENS**

- Enter the following text, and **EXPLAIN CSV**
    - plain text file
    - one row per line
    - column entries separated by commas
    - first row is header data
    - **NEEDS A BLANK LINE AT THE END**
    - **DATA DESCRIBES CATS**
    - Note that the tab is currently `Untitled1`

```
coat,weight,likes_string
calico,2.1,1
black,5.0,0
tabby,3.2,1
```

- **SAVE THE FILE AS `feline_data.csv`**
    - Click on disk icon
    - Enter filename `feline_data.csv`
    - Note that the name in the tab has changed
- **CLOSE THE EDITOR FOR THAT FILE**
- Click on `File` -> `New File` -> `R Script`.
- **EXPLAIN COMMENTS** while entering the code below
    - **COMMENTS ANNOTATE YOUR CODE**: reminders for you, and information for others
    - Comments should **EXPLAIN THE WHY, NOT THE HOW** - the code should be clear enough to explain *how* at task is performed

```
# Script for exploring RStudio

# Load cat data
cats <- read.csv(file = "feline_data.csv")
```

- **EXPLAIN `read.csv()`**
    - `read.csv()` is a **FUNCTION** that reads data from a **CSV-FORMAT FILE** into a variable in `R`
- **SAVE THE SCRIPT**
    - Click on `File` -> `Save`
    - Enter filename `cats` (**EXTENSION IS AUTOMATICALLY APPLIED**)
    - Note the tab name has changed to `cats.R`
- **SHOW THE ENVIRONMENT TAB**
    - This describes all variables in the current `R` environment.
- **ASK: DO YOU SEE THE VARIABLE IN THE ENVIRONMENT?**
    - **NO** - because the code hasn't been executed, only written.
- **RUN THE SCRIPT**
    - Click on `Source`
    - **NOTE THIS RUNS THE WHOLE SCRIPT**
    - **NOTE THE CONSOLE ENTRY**
- Go to the `Environment` tab
    - **NOTE THE DATA WAS LOADED IN THE VARIABLE `cats`**
    - Note that there is a description of the data (3 obs. [rows] of 3 variables [columns])

    - **CLICK ON THE VARIABLE AND NOTE THAT THE TABLE IS NOW VISIBLE** - this is helpful
    - **YOU CANNOT EDIT THE DATA IN THIS TABLE** - you can sort and filter, but not modify the data.
        - This **ENFORCES GOOD PRACTICE: DATA SEPARATION** (compare to Excel).

🟥 Red sticky for a question or issue 🟩 Green sticky if complete

---

SECTION 02: MY FIRST `RSTUDIO` PROJECT

LEARNING OBJECTIVES

- Good practice for `RStudio` project structure
- Load data into an `RStudio` project
- Produce summary statistics of data
- Extract subsets of data
- Plotting data in `R`

PROJECT MANAGEMENT IN RSTUDIO

- `RStudio` **TRIES TO BE HELPFUL** and provides the 'Project' concept

    - Keeps **ALL PROJECT FILES IN A SINGLE DIRECTORY**
    - **INTEGRATES WITH `GIT`**
    - Enables switching between projects within `RStudio`
    - Keeps project histories

- **INTERACTIVE DEMO**

- **CREATE PROJECT**

- Click `File` -> `New Project`

    - Options for how we want to create a project: -brand new in a new working directory
        - turn an existing directory into a project (project gets directory name)
        - or checkout a project from `GitHub` or some other repository

- Click `New Directory`

    - Options for various things we can do in `RStudio`. Here we want `New Project`

- Click `New Project`

    - We are asked for a directory name. **ENTER `swc-r-lesson`**
    - We are asked for a parent directory. **PUT YOURS ON THE DESKTOP; STUDENTS CAN CHOOSE ANYWHERE SENSIBLE**

- Click `Create Project`
- **YOU SHOULD SEE AN EMPTY-ISH `RSTUDIO` WINDOW**
- **INSPECT PROJECT ENVIRONMENT**
- First, **NOTE THE WINDOWS**: editor; environment; files
- **EDITOR** is empty
- **ENVIRONMENT** is empty
- **FILES** shows
  - **CURRENT WORKING DIRECTORY** (see breadcrumb trail)
  - **ONE FILES**: `*.Rproj` - information about your project
- **CREATE DIRECTORIES IN PROJECT**
- **Create directoris called `scripts` and `data`**
  - Click on `New Folder`
  - Enter directory name (`scripts`)
  - Note that the directory now exists in the `Files` tab
  - Do the same for `data/`
- **NOTE THAT WE WILL POPULATE THE DIRECTORIES AS WE GO**

LOADING DATA

- We've already created some cat data manually
  - **THIS IS UNUSUAL** - most data comes in the form of plain text files

START DEMO

- **INSPECT DATA IN FILES WINDOW**
  - Click on filename, and select `View File`
  - Note: **THERE IS NO HEADER** and **THERE ARE NO ROW NAMES**
  - Ask: **IS THIS WELL-FORMATTED DATA?**
  - I happen to know that there is **one row per patient, and the columns are days, in turn, post-treatment**, and **measurements are inflammation levels**
- **WHAT IS THE DATA TYPE**
  - Tabular, with **EACH COLUMN SEPARATED BY A COMMA**, so **CSV**
  - **IN THE CONSOLE** use `read.csv()` to read the data in
  - Note: **IF WE DON'T ASSIGN THE RESULT TO A VARIABLE WE JUST SEE THE DATA**
- **CREATE A NEW SCRIPT**
  - Click the **triangle next to the new document icon**
  - Add the code and **SAVE AS `scripts/inflammation`** (`RStudio` adds the extension)
  - See that the file appears in `Files` window