SPEAKER_NOTES.md - R for Reproducible Scientific Analysis

Speaker Notes for the 2017-01-11 Software Carpentry R for Reproducible Scientific Analysis lesson

TYPE ALL EXAMPLES AS YOU GO. THIS KEEPS THE SPEED SANE, AND ALLOWS YOU TO EXPLAIN EVERY STEP.

START SLIDES WITH `reveal-md slides.md --theme=white`

Introduction to R and RStudio

SLIDE (Why `R` / `RStudio` ?)

- Talk around slide

SLIDE ("But I already know Excel!")

SLIDE (`R` / `RStudio` presentation)

- Live presentation section
- Everyone start up `RStudio`

Summarise windows

- Four (maybe three) subwindows:
  - Interactive `R` console
  - Editor (may be missing on startup - will appear when files are opened)
  - Environment/History
  - Files/Plots/Packages/Help

Create a working directory with version control

- **We're following practices of project management**
  - We'll create a project directory, with `Git` version control
  - Helps ensure data integrity
  - Makes sharing code easier (lab-mates, publication)
  - Easier to recover after a Christmas break
- **Create the new directory LIVE**

- `File->New Project`
- `New Directory`
- `Empty Project`
- Enter sensible name, e.g. `swc-r_reproducible`
- Check box for `Create a git repository`
- `Create project`

 Red sticky for a question or issue  Green sticky if complete

- Describe contents of new folder
  - `.gitignore`
  - `.Rproj`

SLIDE (Best practices)

- Talk around slide

Create directory structure

SLIDE (Creating files/directories)

- **LIVE PRESENTATION**
- **Create subdirectory for data**
  - In `Files` tab, create `data` subdirectory
- **Create new `R` script**
  - `File -> New File -> R script`
  - save in working directory with sensible name, e.g. `swc-script.R`

 Red sticky for a question or issue  Green sticky if complete

- **LIVE VERSION CONTROL EXAMPLE**
- **Show Git tab on right**
- **Stage files**
  - Three files shown (including `.gitignore` and the new script file)
  - Yellow status markers mean they're not in the repository
  - Click check-boxes to stage them

- Note that **we don't version disposable output**

- **Commit files**
  - Click `Commit`
  - Describe new dialogue window
  - Show contents/changes to files
  - Add commit message ("initialise repository") - good commit messages are short and imperative
  - Commit
  - Show commit summary
  - Exit

 Red sticky for a question or issue  Green sticky if complete

SLIDE (Challenge 1)

Run through challenge (5min?) - hint about editing `.gitignore`

- Right-click link on presentation and download to `data`
- Create `graphs` subdirectory in `Files` tab
- Edit `.gitignore` to add `graphs/` folder and save
- Stage `.gitignore` in Git tab
- Commit in Git tab, and add appropriate commit message
- Demo History window for Git

SLIDE (`R` as a calculator)

Interacting with `R`

- **Two ways**
  - Type commands in the console
  - Use the script editor and save the script

- **Console**
  - Output shown here
  - Good for experimentation
  - Commands 'forgotten' when you close a session

- **Script**
  - Keeps record of what you did

- Easier to reproduce and share

Working at the console

- `R` shows a `>` if it is expecting input

```
> 1 + 100
[1] 101
```

- `R` shows `+` if it's waiting for completion (`Esc` to exit)

```
> 1 +
+
```

Working from script file

- Can write same commands in the script file (`1 + 100`)
  - Use `Run` to execute
  - Use `Ctrl-Enter` to execute
  - Output appears in the console
  - Show `#` comments - good practice to comment
  - More examples (order of precedence):

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

- Show `Source` operation: add the following lines to script:

```
# Using R as a calculator script demo
1 + 100
3 + 5 * 2
(3 + 5) * 2
```

- Run script

```
> # Using R as a calculator script demo
> 1 + 100
[1] 101
> 3 + 5 * 2
[1] 13
> (3 + 5) * 2
[1] 16
```

- More examples
  - scientific notation

```
> 1/40
[1] 0.025
> 2/10000
[1] 2e-04
> 5e3
[1] 5000
```

Mathematical functions

- General format: `fn(arg)`
    - autocompletion - example: `factorial(6)`

```
> sin(1)
[1] 0.841471
> log(1)
[1] 0
> log10(10)
[1] 1
> exp(0.5)
[1] 1.648721
```

Comparisons

- Return `TRUE` / `FALSE` `logical` values

```
> 1 == 1
[1] TRUE
> 1 == 2
[1] FALSE
> 1 != 2
[1] TRUE
> 1 < 2
[1] TRUE
> 1 > 2
[1] FALSE
> 1 <= 2
[1] TRUE
> 1 >= 2
[1] FALSE
```

- Computer representation of numbers is approximate: important for comparisons
    - Any physicists/computer scientists in the room?
    - Numbers may not be equal, but be 'the same'
    - Use `all.equal` instead of `==`

```
> all.equal(pi-1e-7, pi)
[1] "Mean relative difference: 3.183099e-08"
> all.equal(pi-1e-8, pi)
[1] TRUE
> pi-1e-8 == pi
[1] FALSE
```

Variables and assignment

- **Variables hold values**, just like in Python
- Two ways to assign variables: `<-` and `=`
  - The `<-` form is more widely used
  - Consistency more important than choice

```
> x <- 1/40
> x
[1] 0.025
> x = 1/40
> x
[1] 0.025
```

- **Look at the Environment tab** automatic updates

```
> x <- 100
```

- Variables can be used as arguments to functions

```
> log(x)
[1] 4.60517
> sqrt(x)
[1] 10
```

- Variables can be used to reassign values to themselves

```
> x
[1] 100
> x <- x + 1
> x
[1] 101
```

SLIDE (Good variable names)

- Talk around slide

SLIDE (MCQ1)

- Pose question

Package management

SLIDE (Package Management)

- See what packages are installed with `installed.packages()`
    - **demo this one**
- Add a new package using `install.packages("packagename")`
    - **demo this one with** `install.packages("ggplot2")`
- Update packages with `update.packages()`
    - **demo this one**
- You can remove a package with `remove.packages("packagename")`
- To make a package available for use, use `library(packagename)`
    - **demo**
    - Note that there are no quotes, this time

```
> ggplot()
Error: could not find function "ggplot"
> library(ggplot2)
Warning message:
package ‘ggplot2’ was built under R version 3.2.3
> ggplot()
Warning message:
In max(vapply(evaled, length, integer(1))) :
  no non-missing arguments to max; returning -Inf
```

SLIDE (Challenge 2)

Solution:

```
install.packages("plyr")
install.packages("gapminder")
install.packages("dplyr")
install.packages("tidyr")
```

Getting help for functions

SLIDE (Functions, and getting help)

- Talk around slide

- Demo: `round(3.14159)`:
  - argument: `3.14159`
  - value: `3`

```
> round(3.14159)
[1] 3
```

SLIDE (Getting help for functions)

- **Carrying on with `round()` from last slide**
- What other arguments can `round()` take?
  - Use `args(fname)`

```
> args(round)
function (x, digits = 0)
NULL
```

- Can use the `digits` argument by naming it, or not (but order matters)

```
> round(3.14159, digits=2)
[1] 3.14
> round(3.14159, 2)
[1] 3.14
```

- **Best practice**: always use the argument name
  - clearer to others
  - if function changes, order may change
  - difficult to remember the purpose of each argument, if not explicit
- **What does a function do?**
  - Use `?fname` or `help(fname)` to get the complete help text
  - Demo: `?round` - go through main points
- **What package is my function in?**
  - (i.e. I can't find it, and don't know what to install)
  - Demo: `??melt` - show that we need `reshape2`
- **Is there a function that does X?**
  - e.g. you know the name of a test, such as Kolmogorov-Smirnov
  - Demo: `help.search("smirnov")`, `?ks.test`

SLIDE (Where can I get more help?)

- Talk around slide

SLIDE (Asking the right questions)

- Talk around slide
- For `dput()` example use `dput(head(iris))`
- Demo `sessionInfo()`

Functions

SLIDE (Functions)

SLIDE (Learning objectives)

- Talk around slide
- **Why functions?**
  - You've already seen the power of functions, for encapsulating complex analyses into simple commands
  - Functions work similarly in `R` as they do in the shell/Python

SLIDE (What is a function?)

- Talk around slide

Defining a function

SLIDE (Defining a function)

- Talk around slide
- **Create a new `R` script file to hold functions**
  - `File -> New File -> R Script`
  - `File -> Save -> functions-lesson.R`
  - Check what's happened in Git tab
- **Write new function in script**
  - Describe parts of function:
  - *prototype* with inputs

- code block/body
- indentation (readability)
- addition, and return statements
- function scope, internal variables (readability)
- assignment of function to variable
- comments (readability)

```
# Returns sum of two inputs
my_sum <- function(a, b) {
  the_sum <- a + b
  return(the_sum)
}
# Converts fahrenheit to Kelvin
fahr_to_kelvin <- function(temp) {
  kelvin <- ((temp - 32) * (5 / 9)) + 273.15
  return(kelvin)
}
```

- **Run the functions**
  - `source` the script
  - tab-completion works!
  - boiling and freezing points

```
> fahr_to_kelvin(32)
[1] 273.15
> fahr_to_kelvin(212)
[1] 373.15
```

SLIDE (Challenge 1)

Solution:

```
kelvin_to_celsius <- function(temp) {
  celsius <- temp - 273.15
  return(celsius)
}
```

SLIDE (Challenge 2)

Solution:

```
fahr_to_celsius <- function(temp) {
  kelvin <- fahr_to_kelvin(temp)
  celsius <- kelvin_to_celsius(kelvin)
  return(celsius)
}
```

INSERTED EXAMPLE

- Just as in Python, we can use `for` loops to apply a function to several values
- Avoids repetition

```
for (i in 32:100) {
  print(fahr_to_celsius(i))
}
```

- Can also apply functions to vectors

```
fahr_to_celsius(32:100)
```

- Also `if` and `if/else` statements, as in Python:

```
if (5 > 1) {
  print("condition is true")
}
```

```
if (5 < 1) {
  print("condition is true")
} else {
  print("condition is false")
}
```

- **COMMIT TO LOCAL GIT REPO**

SLIDE (Testing functions)

- Talk around slide
- **Known good values**
  - water freezes at 32F/0C, boils at 212F/100C

```
> fahr_to_celsius(32)
[1] 0
> fahr_to_celsius(212)
[1] 100
```

- **Known bad values**
  - All values are fair game on Fahrenheit/Celsius, but can't go below 0K

```
> kelvin_to_celsius(-10)
[1] -283.15
```

- **We'd need to modify this for real use!**

SLIDE (Not the best approach…)