1. Basic elements of setting up a package, including naming, directory structure and so forth.

**Naming:** Can contain letters, numbers and periods. It must start with a letter and cannot end with a period. Wickham recommends against using periods to avoid confusion with the names of S3 methods.

**Directory Structure inside the top-level directory**

**package\_name/**

**R/** - contains code in \*.R files

**tests/**

**data/** - contains \*.rda files that are to be distributed with the package

**man/** - contains \*.Rd files for documentation of package, functions and data. These files may be written manually or created using roxygen2.

**inst/** - can contain any files. The contents of this directory will be recursively copied to the top-level directory after installation. So the filenames in this directory should be distinct from the original contents of the top-level directory.

1. Include detailed documentation. What forms of documentation should accompany the code (e.g. help files, readme files, vignettes, comments)? Can you implement roxygen and rmarkdown?

Write *.Rd* files in the *man/* directory of the package

roxygen2: can be used to create \*.Rd files in the man/ directory for documentation. These files are used for documentation of the package, functions, and data. roxygen2 turns specifically formatted comments into *.Rd* files. It can also manage NAMESPACE and the Collate field in DESCRIPTION.

1. Think about how you store data in the package (hint: there are at least two directories in the package where you can store data). Are there situations where JSON or YAML formats are useful for including as part of an R package?

Data may be stored in four directories in a package:

**data/**

Directory to store data that is to be released as part of the package. Data files should not typically exceed 1 MB in size for CRAN release; must be optimally compressed. The data has to be stored in RData files containing a single object with name same as the filename.

Can use the command “*usethis::use\_data(data1, data2)*” to create data1.rda and data2.rda files in the data directory.

**R/sysdata.rda**

To store data that is not available to the users of the package. This is the place to keep data that the functions in the package need. The function objects are created at installation, and these files are not required thereafter.

**inst/extdata**

Directory to store raw data. Raw data may be included as part of a package for examples or vignettes. Are the raw data files typically in JSON/YAML formats?

The contents of *inst/* are recursively copied to the top-level directory after installation. To find a file in inst/ from the code use *system.file()* function.

Example: *system.file( “extdata”, “mydata.csv”, package = “mypackage”)*. Note: the *inst/* directory is omitted from the path specified (arg1) in the *system.file* function call.

**tests/**

It is okay to keep small data files for tests in this directory.

----------------------------------------------------------------------------------------------------------------

1. Use GitHub to help with version control (Sam can get you set up with a private account). Also develop an opinion on version numbering for your own package (e.g. when do you change version numbers, etc)
2. In addition to the core part of the code (i.e. the key algorithm), think about data input and output, plotting, diagnostics (e.g. does the stochastic generator simulate the same statistics as the observed data?), and so forth. I anticipate you’ll be writing quite a few functions for these features.
3. Implement plotting using the default R plots, as well as ggplot. What is the difference in approach? Do you have a preference? Read the ggplot book and form a view in terms of why this is becoming so popular but also why some people do not like it.
4. Think about a coding style guide. Hadley Wickham has developed one, and Google has made some suggested modifications (the link to this is mentioned in either the R Packages book or the Advanced R book). Why do you think Google has departed from Hadley’s style guide and which one do you think is better? Which one should we be using?
5. Develop a condition system that traps errors and provides useful warnings. What does this do not only to code robustness, but also to your coding style?
6. Implement some of the core algorithms in C++ using the RCpp package. There is a chapter on doing this and some basic C++ commands in the Advanced R book.
7. Implement the three different object systems (S3, S4, R6). What is the difference between them? When should you use one over the other?
8. Test for speed – what are different techniques to evaluate code speed, and diagnose computational bottle necks? Does the C++ code improve efficiency in this case? Does it depend on the order (‘O’) of computations – e.g. length of replicates etc. What can you say about how run time scales with length of replicates, vectors and so forth? Does it depend on the nature of the algorithm?
9. Implement unit testing in the package throughout, using the Test That package. What does unit testing do to your coding style? Does it change how you implement functions in R?
10. Read The Pragmatic Programmer. What does implementation of DRY (pg 30) mean for your particular code? What is decoupling? How to you create orthogonality in code? What are some of the key messages in the ‘test to code’ chapter? Why ‘refactor’?