R Basics

In this lesson we are going to go over the very basics of R, cover some basic terminology, talk a little about syntax and point you to resources for getting help.

Lesson Outline:

- Functions and basic syntax
- Packages
- Operators and objects
- Getting help

Lesson Exercises:

- Exercise 2.1

Functions and basic syntax

R is a functional programming language and as such, most everything you do uses a function.

The basic syntax of function follows the form: `function_name(arg1, arg2, ...)`. With the base install, you will gain access to many (3034 functions, to be exact). Some examples:

```r
#Print
print("hello world!")
```

```
## [1] "hello world!"
```

```r
#A sequence
seq(1,10)
```

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

```r
#Random normal numbers
rnorm(100, mean = 10, sd = 2)
```

```
##   [1] 10.920410  9.513012 11.861086 14.863490  8.379840 13.141891  9.446537
##   [8]  8.022032  9.465661  8.772420 13.344523  9.118743 11.891361  9.612997
##  [15]  9.017080 10.226274  8.486031  6.841061 10.848418  8.932206  5.553944
##  [22]  8.752397  9.736154 13.455697  5.119204  9.982638  9.059941  8.795523
##  [29]  6.981665  8.044743  9.222369  9.401621  5.781320 10.480465 11.466891
##  [36]  6.971664  7.482965  8.127270  9.443446  8.659414  6.650244  8.101129
##  [43] 12.354610 12.422165 11.299381  8.624484 10.071697 11.545935  8.203265
##  [50]  9.284808  9.324787 11.729616  9.393644  9.497394 11.158317 10.074143
##  [57]  8.745845  8.455896 11.689187  9.739108  8.052527  7.124063  6.310835
##  [64] 10.877433 13.526127  9.810944  6.981618  7.402550  9.043106  8.284713
##  [71] 13.555172  9.792659 10.878405  7.276007 10.122262 12.079535  7.506201
##  [78] 11.190019 10.375745 11.711439  8.880767 11.818784 10.420440 10.057292
##  [85] 10.314723  8.253971 10.394187  8.835908  7.187634 10.845006 11.841143
##  [92]  9.217214 10.086156  6.801660 12.951945 10.548676  8.985280 11.155482
##  [99] 11.262077 13.060215
```

```r
#Mean
mean(rnorm(100))
```

```
## [1] 0.01657787
```

```
#Sum
sum(rnorm(100))
```

```
## [1] 0.3787673
```

A few side notes

There are several other characters that commonly show up in R code. These are:

```
# - comments
() - wraps function arguments and order of operations
[] - indexing
{} - grouping code
```

The `#` indicates a comment. You can put whatever else you'd like after this, but on the same line as the `#`. R will not evaluate it. Multiple `#####`, are still just seen as a comment. When commenting your code, err on the side of too much! Also, you will see `()`, `[]`, and `{}` used in R code. The `()` indicates a function (almost always), the `[]` indicates indexing (grabbing values by the location), and the `{}` groups code that is meant to be run together and is usually used when programming functions in R.

Packages

The base install of R is quite powerful, but you will soon have a need or desire to go beyond this. Packages provide this ability. They are a standardized method for extending R with new methods, techniques, and programming functionality. There is a lot to say about packages regarding finding them, using them, etc., but for now let's focus just on the basics.

CRAN

One of the reasons for R's popularity is CRAN, The Comprehensive R Archive Network. This is where you download R and also where most will gain access to packages (there are other places, but that is for later). Not much else to say about this now other than to be aware of it. As of 2019-11-26, there are 15242 packages on CRAN!

Installing packages

When a package gets installed, that means the source is downloaded and put into your library. A default library location is set for you so no need to worry about that. In fact on Windows most of this is pretty automatic. Let's give it a shot.

```
#Installing Packages from CRAN
#Install dplyr and ggplot2
install.packages("ggplot2")
install.packages("dplyr")

#You can also put more than one in like
install.packages(c("quickmapr", "formatR"))
```

Now a couple of words of warning for library locations on EPA windows machines. If R was installed correctly, your library should reside in something like `C:/Program Files/R/R-3.6.1/library`. If you type `.libPaths()` and get a vastly different result (e.g. `Blah\Blah\Net MyDocuments\Blah`), then you are going to have some problems. We can fix this by adding a file, `Renviron.site` to `C:\Program Files\R\R-3.6.1\etc`. That file should have the following lines in it

```
HOME=C:/Program Files/R/R-3.6.1
R_USER=C:/Program Files/R/R-3.6.1
```

On your next start up of R, the library path should look better!

Using packages

One source of confusion that many have is when they cannot access a package that they just installed. This is because getting to this point requires an extra step, loading (or attaching) the package.

```r
#Loading packages into your library
#Add libraries to your R Session
library("ggplot2")
library("dplyr")

#You can also access functions without loading by using package::function
dplyr::mutate
```

```
## function (.data, ...)
## {
##     UseMethod("mutate")
## }
## <bytecode: 0x00000233bbc3ea98>
## <environment: namespace:dplyr>
```

You will often see people use `require()` to load a package. It is better form to not do this. For a more detailed explanation of why `library()` and not `require()` see Yihui Xie's post on the subject

And now for a little pedantry. You will often hear people use the terms "library" and "package" interchangeably. This is not correct. A package is what is submitted to CRAN, it is what contains a group of functions that address a common problem, and it is what has allowed R to expand. A library is, more or less, where your packages are stored. You have a path to that library and this is where R puts new packages that you install (e.g. via `install.packages()`). These two terms are related, but most certainly different. Apologies up front if I slip and use one when I actually mean the other...

Operators and objects

As I mentioned above, the console and using R interactively is very powerful. We will do this quite a bit. Let's spend a little time playing around in the console and learn a few new functions.

R can be used as a calculator and a way to compare values. Some examples of the basic operators:

```r
#A really powerful calculator!
1 + 1 #Add
```

```
## [1] 2
```

```r
10 - 4 #Subtract
```

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

```r
3 * 2 #Multiply
```

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

```r
3 ^ 3 #Exponents
```

```
## [1] 27
```

```r
100 / 10 #Divide
```

```
## [1] 10
```

```r
5 %% 2 #Modulus
```

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

```r
5 > 2 #Greater than
```

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

```r
4 < 5 #Less than
```

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

```r
5 <= 5 #Less than or equal
```

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

```r
8 >= 2 #Greater than or equal
```

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

```r
2 == 2 #Equality: notice that it is TWO equal signs!
```

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

```r
5 != 7 #Not Equals
```

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

That's neat, but so what…

Well, it could be interesting to do something with those values and save them for re-use. We can do that with objects (everything in R is an object) and use the assignment operator, `<-`. Know that object names cannot start with a number, contain spaces, or (most) special characters. Underscores and periods are allowed.

NOTE: If you have experience with other, object-oriented languages, then just be aware that R objects, at least the general use of the term, are different.

```r
#Numeric assignment
x <- 5
x
```

```
## [1] 5
```

```r
y <- x + 1
y
```

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

```r
z <- x + y
z
```

```
## [1] 11
```

```r
#Character
a <- "Bob"
a
```

```
## [1] "Bob"
```

```r
b <- "Sue"
b
```

```
## [1] "Sue"
```

```r
a2 <- "Larry"
a2
```

```
## [1] "Larry"
```

Now that we have a little experience working in the console and creating objects with `<-`, we might want to be able to do some additional things to navigate around, look at these objects etc. You can do a lot of this directly in RStudio in the Environment, History pane which is likely in the upper right corner of the window. Alternatively, you can explore your current environment via the console. Some functions that you might find useful for working with your R workspace:

```r
#List all objects in current workspace
ls()
ls(pattern = "a")

#Remove an object
rm(x)

#Save your workspace
#Saves the whole thing to a file called lesson2.RData
save.image("lesson2.RData")
#Saves just the a and y objects to a file called lesson2_ay.RData
save(a, y, file = "lesson2_ay.RData")
```

This is probably a good spot to bring up quotes vs no quotes around arguments in a function. This is a very common stumbling block. The general rule is that no quotes are used only when referring to an object that currently exists. Quotes are used in all other cases. For instance in `save(a, y, file = "lesson2_ay.RData")` the objects `a` and `y` are not quoted because they are objects in the workspace. `file` is an argument of save and argument names are never quoted. We quote the name of the file "lesson2_ay.RData" because it is not an R object but the name of a file to be created. You will likely still have some issues with this. My recommendation is to think about if it is an object in your R workspace or not. If so, no quotes! This isn't foolproof, but works well most of the time.

Next thing you might want to do is navigate around your files and directories. While you can do this directly from the console, it is going to be better practice to mostly use RStudio projects to manage your folders, working directory etc. You can also navigate using the Files, etc. pane. ### A quick word about assignment We have now seen how to assign object in R using the assignment operator, `<-`. Assignment can also be done with `=`. I suggest that you don't use this, not for "correct" syntax reasons, but for readability and code style reasons. Use `<-` for assignment (fun note, `->` also works but the object it is assigned to lives on the right side of the operator). Reserve `=` for assigning arguments inside of functions (e.g. `rnorm(10, mean = 5)`).

Getting help

Being able to find help and interpret that help is probably one of the most important skills for learning a new language. R is no different. Help on functions and packages can be accessed directly from R, can be found on CRAN and other official R resources, searched on Google, found on StackOverflow, or from any number of fantastic online resources. I will cover a few of these here.

Help from the console

Getting help from the console is straightforward and can be done numerous ways.

```
#Using the help command/shortcut
#When you know the name of a function
help("print") #Help on the print command
?print #Help on the print command using the `?` shortcut

#When you know the name of the package
help(package="dplyr") #Help on the package `dplyr`

#Don't know the exact name or just part of it
apropos("print") #Returns all available functions with "print" in the name
??print #Shortcut, but also searches demos and vignettes in a formatted page
```

Official R Resources

In addition to help from within R itself, CRAN and the R-Project have many resources available for support. Two of the most notable are the mailing lists and the task views.

- R Help Mailing List: The main mailing list for R help. Can be a bit daunting and some (although not most) senior folks can be, um, curmudgeonly…
- R-sig-ecology: A special interest group for use of R in ecology. Less daunting the the main help with participation from some big names in ecological modelling and statistics (e.g., Ben Bolker, Gavin Simpson, and Phil Dixon). One of the moderators is great, the other is a bit of a jerk (it's me).
- Environmetrics Task View: Task views are great in that they provide an annotated list of packages relevant to a particular field. This one is maintained by Gavin Simpson and has great info on packages relevant to much of the work at EPA.
- Spatial Analysis Task View: One I use a lot that lists all the relevant packages for spatial analysis, GIS, and Remote Sensing in R.

Google and StackOverflow

While the resources already mentioned are useful, often the quickest way is to just turn to Google. However, a search for "R" is a bit challenging. A few ways around this. Google works great if you search for a given package or function name. You can search for mailing lists directly (i.e. "R-sig-geo"). An R specific search tool, RSeek.org, has been created to facilitate this.

One specific resource that I use quite a bit is StackOverflow with the 'r' tag. StackOverflow is a discussion forum for all things related to programming. You can then use this tag and the search functions in StackOverflow and find answers to almost anything you can think of.

Other Resources

As I mention earlier, there are TOO many resources to mention and everyone has their favorites. Below are just a few that I like.

- R For Data Science: Another book by Hadley and Garrett Grolemund. First reference I suggest for new (and experienced) R Learners, especially for all things Tidy.
- Advanced R: Web home of Hadley Wickham's new book. Gets into more advanced topics, but also covers the basics in a great way.
- R For Cats: Basic introduction site, meant to be a gentle and light-hearted introduction.
- CRAN Cheatsheets: A good cheat sheet from the official source

- RStudio Cheatsheets: Additional cheat sheets from RStudio. I am especially fond of the data wrangling one.

Exercise 2.1

We should still have our `nla_analysis.R` file open. We will be working with this as we go through the rest of the excercises.

Take a look at this file and with the person sitting next to you find the following:

1. Find the `read_csv()` function. What lines is it on? What is the argument?
2. Now find the lines on which you think we install packages and load libraries. There is the fancy way (lines 19-24) and a straight up way (lines 29-32). Talk through in your own words what each of these is doing
3. Add a line of code after line 24 to install the package `lubridate`. Add a line after line 33 to load `lubridate`.
4. Bring up the package level help for the `lubridate` package. What does this package do?