Introduction to Objects, Functions, & Workspace

Getting your head wrapped around some basic concepts in R will make working with R go much more smoothly. Everything in R is either an **object** or a **function**. (Technically, a function is itself an object, but I think creating this artificial dichotomy is the easiest way to understand R.)

Objects are exactly what you might expect – things that R works with. An object may be a number, a word, a set of data, etc. Related to this is the important concept of a **workspace**. When you start R it opens a workspace and all the objects you use during your session are stored in that workspace. Unless you save the workspace when you finish your R session, the next time you open R those objects are no longer present. If you have saved your workspace you can re-load it after you start R and those object will then become available.

Functions do things to objects. You know you are working with a function when it has parentheses at the end of its name. Often you put into the parentheses the object you wish the function to act on e.g. `mean(heights)` will calculate the mean of the object 'heights'. Options for how the function runs may also be included inside the parentheses. However sometimes no object or options need to be specified, e.g. `ls()` will give you a list of all objects in your current workspace. R has many built-in functions that you can access from any workspace. There are also many packages that provide extra functions that are available once the package is installed and its library loaded into your current workspace. Finally, you can write your own functions, but like the objects you create they will only be available in the workspace in which they've been created.

Short of assigning a single number to a variable name, everything you will do in R will involve functions. Understand what a function is and you are half way there.

Some Starting Tips

Two very handy keys when using R interactively (i.e. in the console) are ? and ↑. We will use these over and over again in the next few days. The up arrow recalls your last submitted command to R – this is extremely handy when you've made a typo! (R is particular – the smallest typo will give you an error and you will have to try again.)

The question mark is used to bring up the help text for any function that follows it. e.g. `?mean()` will bring up the help page for the `mean()` function. In RStudio you can readily access this using the help page and its search window of course, but `?` will work no matter what R interface you are using. We'll look at the help in R in depth a little later.

Objects

Assigning Values to Objects

The simplest object is a single number that your assign a name (symbol) to.

```
obj1 <- 32
```

Note the assignment operator <- Get used to using this to assign values; while it may seem awkward at first you will quickly become used to it. It is true that in most cases R will let you use the more traditional = without error, but it is not standard syntax for R and it does not always work.

Types of Objects

- **Vectors**:
  - Basic types ('atomic' vectors in R)
    - 3 types you are likely to use: logical, numeric, character

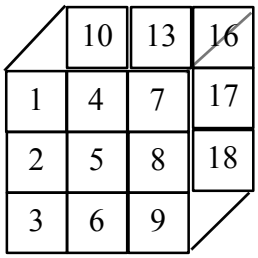
1	2	3

Note that in R a scalar (single number or character string) is considered a vector of length 1.

It is possible to have a vector of length 0 (and this is not the same as the special object NULL).

- **Matrix** (2-dimensions)**, Array** (>2 dimensions)– multidimensional generalization of a vector

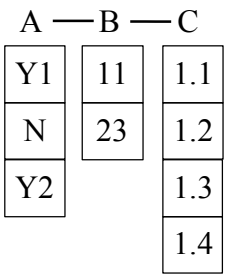
1	4	7
2	5	8
3	6	9



- **Data Frame** - similar to a matrix but designed specifically for data with one row per observation and both numerical and categorical variables; all variables must have same number of rows

	V1	V2
	A	11
	B	23
	C	45

- **List** – general form of a vector that can different types of vectors
- **Factor** – specialized vector for handling categorical data (often variable in data frame)



Data frames are what you will most commonly work with – when you read in data (see below) it will automatically make it a data frame.

Object Attributes:

- names – if present, label elements of vector
- dim – dimensions of a matrix or array
- dimnames – if present, labels for each dimension of an array
- class – for a simple vector, the class will be whether elements are numerical, character, etc. but many functions produced highly specific classes for lists and you may need to know the class of an object to see if is of the appropriate class for a particular function

» Data types: You might think all pieces of data are alike, but they are not and you can make your R experience much easier by recognizing the different types of data and how they are stored in R.

Creating Objects with >1 Value

To create a vector of length >1 in R, the first thing we have to do is use a function to put together the values. This function is **`c()`** or the concatenate function. Everything within the parentheses are put together and can then me assigned to an object.

```
numbers <- c(1,5,9,3,7)
letters <- c("a", "b", "d")
words <- c("cat", "hat", "thing1", "thing2")
```

A 2-dimensional matrix (i.e. a table of numbers) can be created using the matrix() function.

```
mat1 <- matrix(c(1,5,9,8,12,10), nrow=3, ncol=2)
```

Getting Information about Objects

In these simple objects that we create ourselves it is pretty clear exactly what the object is composed of. However, when more complex functions are saved as objects, or when we've read in data files, the attributes of the object are not always so obvious. Since the attributes will affect the way objects are handled by functions, it is key that you ensure you know exactly what your objects look like.

As an example, one 'gotcha' I see regularly for people, is with factors in data frames. A numerically coded factor must be specified as such or your analysis will be incorrect!

» Memorize the following function (structure): `str()` and use it often! This will give you all the information about an object. For a specific attribute you can also use the functions: `class()`, `dim()`, `names()`, `dimnames()`.

Finding your way around in the Workspace

Also useful is being able to see what objects are in your workspace and what directory you are working in!

`ls()` – 'list' - the list function with no options will list all objects in your current workspace (you can add options to list only specific objects)

`getwd()` – 'get working directory' – this function will provide the full path of the directory (folder) of your computer that you are currently working in. Useful when you are reading in files – if your file is not in the current working directory, then the full path must be specified in its name (see below). In the Mac R interface and in RStudio, the working directory is also given at the top of the console window.

`setwd("directory_name")` – 'set working directory' – place the name of the directory you wish to use inside the parentheses as an option. If the folder you want is in your current working directory you need only give the name of the folder; if it is not you will need to give the full path name. In RStudio (and most R interfaces) it is easier to use the menu to choose your directory interactively.

The best practice to get into when doing analyses in R is to set up a directory in your documents folder for that analysis. So, for example, you might create directory called "R Boot Camp" for this week, put all the data files you download for the course into that directory and then ensure you are in that directory when you are working in R. This way, your saved script files, data files, graphs, and data output will all be together in one place.

RStudio has made keeping track of items in the workspace very much easier – I now rarely use the ls() or getwd() functions – the Environment window pane of RStudio tells you exactly what is currently in your workspace. This has the added benefit of making it easy to check that any objects you create actually were created and are what you expected. In the most recent versions of RStudio you can even see what variables are in your data.frame.

RStudio also allows you to create **Projects** – this is a very convenient way to keep all of your files and workspace objects together – opening the Project also puts you into the right folder/directory and loads your recent workspace.

Operators are Functions

The standard mathematical functions (+, -, / , *, >, <) are actually simple functions that do not have the standard function format. Two operators you may not be familiar with, but will need are ! (not) and == (equal to).

Accessing Components of your Data (Indexing)

Fundamental to learning R is understanding how your data are stored so that you can easily access the values you want. R provides great flexibility in this regard, but wrapping your head around the mechanics can take a bit of practice.

To explicitly pull out values, we use the index operator (a function with a non-standard function format as with mathematical operators), the square bracket character **[**

```
> letters[3]
[1] "d"
> words[1]
[1] "cat"
```

For a matrix, which has 2 dimensions, the form is [row# , col#]

```
> mat1[2,2]
[1] 12
To transpose a matrix, use the function t(), e.g.
> mat1
     [,1] [,2]
[1,]    1    8
[2,]    5   12
[3,]    9   10
> t(mat1)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    8   12   10
```

Because **[** is actually a function, we can place other functions inside of it and use it to find particular values for us. We will see this more in future.

As a quick example, consider our character vector, "words" and imagine we want to find all values in that vector that contain 'thing'. We can use the grep function (you may be familiar with this type of regular expression matching from other programs).

grep() takes a pattern and a character vector as arguments and will search for text elements in an object that match some expression, e.g.,

```
> words.subset <- words[grep("thing*", words)]
> words.subset
[1] "thing1" "thing2"
```

Data Input and Output

Be sure to write and read files to/from a document folder and NOT your desktop!

`read.table("filename")` – a general form of the function that allows you to specify what character separates your data values. Place the filename with your data as the first option inside the parentheses. Other commonly used options are `header=` (TRUE if you the first row of your file is column names; =FALSE if not) and `sep = " "` (place the separating character inside the quotes).

If you assign this function result to an object, it will be a **data.frame**

`read.csv("filename")` – a more specific function that defaults to comma separated data; what you are most likely to use as you read in data files that were saved as '.csv' files from a spreadsheet

`write.table(object, "filename")` – write the data.frame called object to the text file 'filename'. Defaults to space-delimited but allows you to specify delimiter with option sep = " "

`write.csv(object, "filename")` – as above, but default delimiter is comma

Filenames

Note that if your file is not in your current working directory, you will need to include the full path to the filename to read in the data. To avoid this, we typically ensure that are working directory is set to be the directory that contains the file.

If you really hate having to type in your filename, you can replace "filename" with `file.choose()` to bring up a dialogue window and browse to your file.

Getting Help in R

There are a variety of ways to access help directly from within R.

`?t.test` *or* `help(t.test)` will bring up documentation for that function.

`??anova` *or* `help.search("anova")` will search help documentation for the term inside the quotations.

`apropos("t.test")` will bring up all functions that contain have "test" in their name.

`example(t.test)` will run an example using this function.

`RSiteSearch("repeated measures")` will search for the "repeated measures" on the R website and return search results in browser window.

Interpreting the R help files can be a bit puzzling at first. They are all set out in a similar format:

- Description
- Usage – often the only section you need, it lays out the syntax of the function, its arguments and their default values
- Arguments – more details about each argument
- Details – more information about how the function can be used
- Value – what type of object the function returns
- References

- See Also – related functions – often helpful in pointing out a different function that might better suit your task
- Examples – these are meant to be copied and pasted into the R console and run – they will make little sense until you do so!

We will go through plenty of help files, so you'll get lots of practice making sense of them!

CRAN is your Friend

One of the best things about R is the large number of resources available online.

There are many, many resources on the R project website (cran.r-project.org). Take advantage of them!

Another of the best things about R is the large number of packages available for carrying out a variety of statistics. R comes loaded with all the functions we've talked about so far and many more, including most of the basic statistical tests. But as open-source software many people have expanded R by putting together packages that supply useful data analysis tools and specialized statistics.

The flip-side of all this is that the amount of material is overwhelming and it is often difficult to know which package you want. Also, because many different people contribute there is often inconsistency in how things are done from one package to the next. However, these downsides are small relative to the large benefit of having so many great tools and a large, helpful online community.

Several other websites provide useful R information and here are some of the ones I've used. (Links to these and others are available on the workshop website.)

- Quick-R (www.statmethods.net). The author of this web site has also published a book that I've personally found very helpful for learning R. The book is *R in Action* published by Manning; the 2nd edition published in 2015 is even better than the 1st. It is available in print and pdf form at https://www.manning.com/books/r-in-action-second-edition
- RStudio (www.rstudio.com/resources) has developed a large number of resources over the past few years, including webinars and helpful blog articles.
- Dolph Schluter's course web page for Quantitative Methods in Ecology and Evolution has lots of helpful info, especially useful is his R tips page (http://www.zoology.ubc.ca/~schluter/R/)
- Paul Johnson's R Tips (http://pj.freefaculty.org/R/Rtips.html) – lots of example code here
- R-bloggers (www.r-bloggers.com) – a central site that has articles produced folk who blog about using R. (Bet you didn't know that such people existed! There are a lot of them. Sometimes helpful.)
- DataCamp (https://www.datacamp.com/home) - this has online courses, including a large number for various aspects of R. Their "Intro to R" course is available for free, others require a monthly subscription.

Working with a Data Frame

Accessing Data Frame Variables

The indexing operator we've already talked about will always work. So you could, for example (assuming you called your data object 'exer1') select the variable 'Response' by typing: exer1[,4] (reminder – nothing before the comma means *all* rows, after the comma we indicate we want only column 4).

There is also a shortcut operator for data frames that lets you more readily select the columns by their name (reminder – use `names()` if you forget what the names are), **$** (the dollar sign key on your keyboard; you will hear me call this 'string' – proof positive that I am an old geek). Thus we can select the response column using: `exer1$Response`.

For completeness, I'll note here that there is a function `attach()` which 'attaches' your data frame so that you can refer to the variable names without having to type the data.frame name. Keeping our same example above, if you use `attach(exer1)` you could then use `mean(Response)` to get the mean value of the response variable. (Without using attach, the code would be `mean(exer1$Response)`. When you are finished with that data frame use the command detach() to remove it again. BE CAREFUL. If you have the same variable name in two data frames and attach the second without detaching the first, then the first variable will be overwritten by the second. Use `detach(exer1)` to detach data frame when you are done. **The use of attach() is strongly discouraged; I never use it myself and will not use it during our workshop.**

A final option is the function `with()`. e.g. `with(exer1, mean(Response))`. This is fine for this type of use, but note that if you assign any values to objects within the `with()` function, those objects will not be available to you outside of the function. In other words, `with(exer1, a<-mean(Response))` will *not* allow you to subsequently use object `a`.

Let's try a run-through with a real data set. We will use the dataset file 'RData.csv'.

Read in the data file and save it to the object dat:

```
dat <- read.csv("RData.csv")
```

Checking Data

It is a very good idea to use `str()` after reading in data to make sure you have what you expected.

`head(dat)` will list the first 6 lines of the data for you – another way to check that the data look as you expected. The function `tail()` will show you the last 6 lines of the data.

I often use `dim()` or `length()` (use the latter for a vector) to quickly check that I have the number of rows and columns I was expecting.

Modifying Data Frames

To create a new data frame with only a subset of variables:

```
part1 <- dat[,1:4]
```

```
part1a <- dat[,c(1:4,9,14)]
```
– for non-contiguous columns, provide a vector of columns you want

or `part1 <- data.frame(dat$Species, dat$trtmt, dat$fish, dat$parasite)`

If you'd like to give your variables different names, you can do that here also:

```
part1 <- data.frame("spp" = dat$Species, "trt" = dat$trtmt, "fish" =
 dat$fish, "parasite" = dat$parasite))
```

To combine vectors or data.frames of the same length:

```
part2 <- dat[,c(6:8,10:12)]
```

```
tog <- data.frame(part1, part2)
```

Since part1 and part2 combined contain all the original columns except two, we could get this same data frame by removing those two columns from the original data:

```
tog1 <- dat[,c(-5,-13)]
```

We can also combine vectors using the more general `cbind()` function. But note the difference when you used `cbind()` instead of `data.frame()` to create 'part1' above.

To add more observations to a data set use `rbind()`:

```
samp1 <- dat[1:30,]
```

```
samp2 <- dat[40:74,]
```

```
combined <- rbind(samp1,samp2)
```

Note that `rbind()` only works if both data.frames or matrices have the same column names.

To select observations based on a particular criterion, if you have a data frame you can simply set your criteria within the indexing function:

```
bulls <- part1a[part1a$spp == "bullfrog", ]
bulls.fish <- part1a[part1a$spp == "bullfrog" & part1a$trt == "fish", ]
```

In a matrix (or a data.frame) we can use `which()`:

```
frogs2 <-part1b[which(part1b[,1] == 3 | part1b[,1] == "2"), ]
```

Note that because we created part1b using `cbind()`, it is a matrix and the factor levels were converted to integers, so now that is what we have to select with.

The `subset()` function provides an easy way to select observations (subset option) and variables (select option):

```
newdat <- subset(dat, subset = flicksPOST > 0, select=c("Species", "trtmt",
     "flicksPRE",'flicksPOST'))
```

Have No Fear

Trial and error is one of the best ways to figure out how to make R do what you want it to. R will not break! You will have your original data file in its spreadsheet or database and it will not be affected by your R code (presuming you do not overwrite it, but you would never do that, right? Save any data from R into a new file!). The worst that will happen is you might have R 'hang' and have to force it to quit (this happens very rarely in my experience). But mostly you will just learn lots of R error messages.

Practice. The package "MASS" comes pre-installed with base R, so you should be able to load the library from your packages window. Once you've loaded the library, load the bacteria data:

```
data(bacteria)   # Remember to check the structure of the data!
```

Create a new data set includes only the treatments 'drug' and 'drug+' and includes the variables ID, week, y, and trt (in that order).

Write this new data set to a csv file on your computer and make sure you can open it successfully in spreadsheet or text editor.

Go onto the CRAN website, go to the "Task Views" and look at any of the task views that is of interest to you. Again, following your interests/research needs, find a relevant package then follow through on the package details. Describe one of the functions in that package – what does it do and how does the function work (i.e. what arguments does it need and what values does it return)?