# Stat 33A - Lecture 1

January 27, 2020

Lecture notes will usually be posted to bCourses by the end of the day.

Useful links:

- RWeekly, news about R
- RStudio Cheat Sheets

## Setting Up

R is an interactive language designed for statistical computing.

RStudio (a separate piece of software) is an integrated development environment (IDE) for R.

We'll use RStudio in this course. RStudio bundles in R, so you only need to install RStudio.

This week's lab will cover setup, but if you want to get ahead, you can download and install RStudio Desktop (free edition) from:

https://rstudio.com/products/rstudio/download/

## R Expressions

The commands we type and send to R are called **expressions**:

```r
2 * 20
```

```
## [1] 40
```

## The R Prompt

R has a Read-Eval-Print Loop (REPL):

1. Type an expression at the R prompt and hit the enter key.
2. R reads the expression.
3. R evaluates the expression to compute a result.
4. R prints the result in the console.
5. R loops back to waiting for you to enter an expression.

## R Notebooks

In this course, we'll mostly use R Markdown Notebooks rather than using the R prompt directly. Most of the lecture notes and assignments will be R Markdown Notebooks.

*The notes in this section are provided by RStudio.*

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

```r
2 + 2
```

```
## [1] 4
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

## Order of Operations

Arithmetic in R follows an **order of operations**, similar to the order of operations you probably learned in a math class. Operations are computed from from top to bottom:

1. Parentheses `( )`
2. Exponentiation `^`
3. Multiplication `*` and Division `/`
4. Addition `+` and Subtraction `-`

You can test this out in the R prompt or notebook to get a feel for it:

```r
5 * 6 - 3
```

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

R has many more operations besides the ones listed above.

You can see R's complete order of operations with the command:

```
?Syntax
```

We'll learn about some of these operations later on.

## Attendance Question

Go to

`https://www.yellkey.com/could`

to answer. You have until approximately 5pm today.

## Using Functions, Part 1

In R (and most programming languages), a **function** is a reusable command that computes something.

Again, this idea is similar (but not identical) to the functions you probably learned about in math class.

If we want the sine of 3, we can write:

```r
sin(3)
```

```
## [1] 0.14112
```

When we use a function to compute something, we usually say we "called" the function.

R has many built in functions for doing math, statistics, and other computing tasks.

```r
sqrt(5)
```

```
## [1] 2.236068
```

```r
sum(1, 2, 3)
```

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

```r
3 + 4
```

```
## [1] 7
```

## Getting Help

R has built-in help files!

You can use the `?` command to get help with a specific function:

```r
?sin
```

The `?` command should be your first stop when you learn a new function.

To access the help for an arithmetic operator, you need to put the name in single or double quotes:

```r
?"+"
```

Using quotes this way works for ordinary functions as well:

```r
?"sin"
```

## Strings

Sometimes we'll need to treat text as data. In programming languages, a sequence of characters (usually textual) is called a **string**.

We already saw how to create a string in R: surround characters with single or double quotes.

```r
"hello class"
```

```
## [1] "hello class"
```

Note that the quote at the beginning needs to match the quote at the end.

## Variables

You can save the result of a computation by assigning it to a **variable**. A variable is a name for a value.

Use `=` or `<-` to assign a value to a variable:

```r
x <- 35
```

Either is okay, but choose one and be consistent!

Variable names can contain letters or numbers, but can't begin with a number:

```r
ducks4 = 3
```

We use use variables to: * Avoid redundant computations by storing results. * Write general expressions, such as `a*x + b` * Break code into small steps, so that it's easier to test and understand.

```r
sin(sqrt(2)) + 3 ^ 2
```

```
## [1] 9.987766
```

```r
sqrt2 = sqrt(2)
sine = sin(sqrt2)
```

**Copy-on-Write**

R's variables are **copy-on-write**.

That is, if we assign `x` to `y`:

```
x = 3
y = x
```

And then change `x`:

```
x = 12
```

Then `y` remains unchanged:

```
y
```

```
## [1] 3
```

Originally, `x` and `y` referred to the same value in memory.

When we changed `x` (a "write"), R automatically copied the original value so that `y` remained the same.

## Using Functions, Part 2

The values we provide as input to a function are called **arguments**. Some functions accept exactly one argument:

```
tan(3)
```

```
## [1] -0.1425465
```

Some functions do not accept any arguments at all. For instance, the `sessionInfo()` function, which returns information about your R software:

```
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Arch Linux
##
## Matrix products: default
## BLAS:   /usr/lib/libopenblasp-r0.3.7.so
## LAPACK: /usr/lib/liblapack.so.3.9.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.6.2 magrittr_1.5    tools_3.6.2     htmltools_0.4.0
##  [5] yaml_2.2.0      Rcpp_1.0.3      stringi_1.4.5   rmarkdown_2.1
##  [9] knitr_1.27      stringr_1.4.0   xfun_0.12       digest_0.6.23
## [13] rlang_0.4.3     evaluate_0.14
```

Some functions accept multiple arguments:

```r
# Comments on code start with '#'
log(12, 2) # logarithm of 12, using base 2
```

```
## [1] 3.584963
```

Some functions accept any number of arguments:

```r
sum(1, 2, 3.1)
```

```
## [1] 6.1
```

Each argument gets assigned to a specific parameter of the function.