Introduction to

Jason Thomas

November 6th, 2018

In collaboration with the WHO VA Reference Group

Goals for this Session

Basic R Commands

Help Files

Data Structures

Functions

Installing Packages

Motivation

- `R` environment is an integrated suite of software
  - data manipulation, statistical analysis, *graphics*, & programming
  - open source
  - flexibility may slow us down, but remedied by hooks into Java/C
  - Shiny Apps
- Latest & Greatest
  - `R` CRAN Packages
  - `openVA`
  - `CrossVA`
- Tools for Reports in the RStudio Integrated Development Environment (IDE)
  - rmarkdown & knitr
  - integrated with GitHub, version control, & Shiny

Goals for this Session

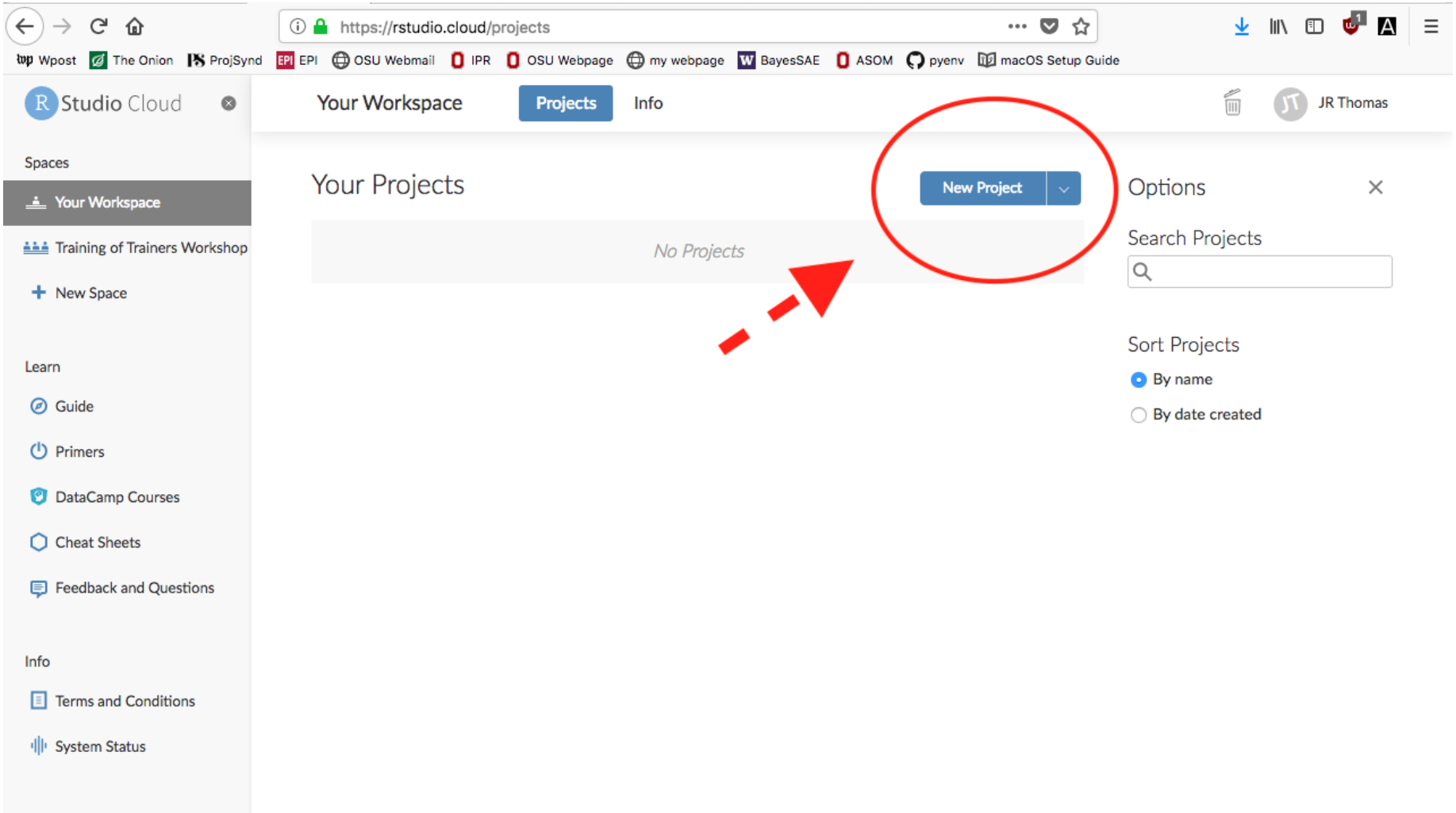
After today we will know. . .

- basic `R` commands and tools for creating and loading data
- how to access and search help files & documentation
- data structures in `R`
- create our own functions
- how to install `R` packages
  - wrestle with `rJava`

Basic R Commands

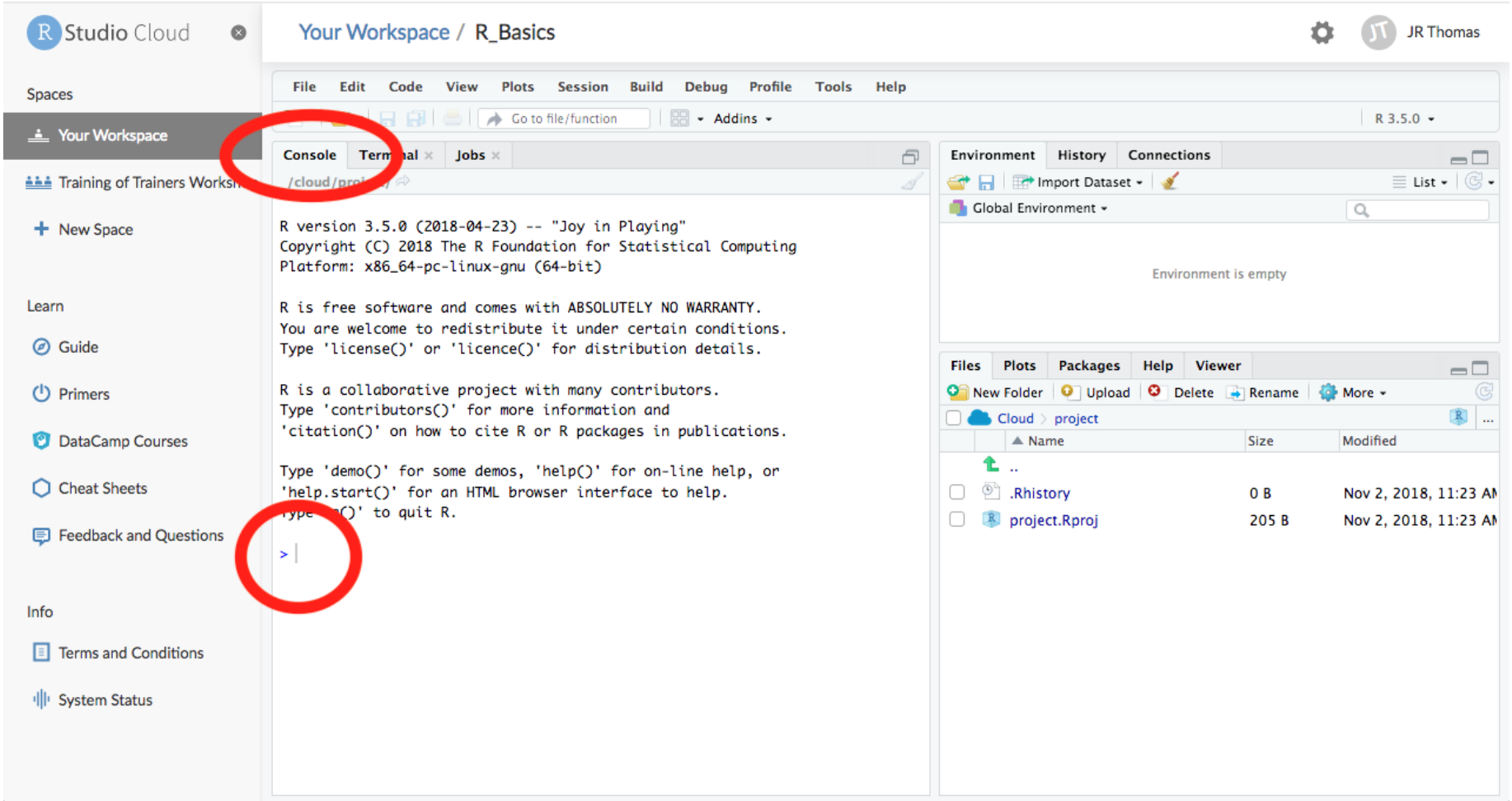
R Studio Cloud: new project

Open RStudio Cloud and create a new project called **R_basics**.



R Studio Cloud: console

In **console** window, enter commands at the **prompt** >



Basic R Commands: set working directory

Comments begin with # and R output begins with ##

```
> getwd()          ## print working directory (wd)
```

```
## [1] "/Users/jason/GitHub/ToT_Workshop_Materials/Jason/code
```

```
> dir()            ## print contents of wd
```

```
## [1] "interva.R"          "introR_data1.csv" "template.R"
```

```
> dir("../")       ## print contents of folder above wd
```

```
## [1] "codes"      "data"       "dependent" "logos"
## [5] "tex"
```

```
> setwd("../")     ## set wd to parent folder
> dir()
```

```
## [1] "codes"      "data"       "dependent" "logos"
## [5] "tex"
```

Basic R Commands: R is a fancy calculator

```
> 2 + 4 - (3 * 5)
## [1] -9
> exp(1)
## [1] 2.718282
> log(exp(5))
## [1] 5
> 3^2
## [1] 9
```

```
> 4 == 4.000001
## [1] FALSE
> 4 != 4.000001
## [1] TRUE
> 3 > 2
## [1] TRUE
> 3 <= 2
## [1] FALSE
```

Basic R Commands: creating objects

R Syntax: objectName <- objectValue

- Create a new object called x

```
> x <- 1
```

- Print the object's value by entering the name

```
> x
```

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

- Names are case-sensitive

```
> X <- 2
> x == X
```

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

Basic R Commands: ls() & rm() commands

Use `ls()` to list objects in R's memory and `rm()` to remove an object.

```
> ls()
```

```
## [1] "x" "X"
```

```
> x1 <- 8.3; x_2 <- 4
> x.3 <- 5
> ls()
```

```
## [1] "x"   "X"
## [3] "x_2" "x.3"
## [5] "x1"
```

```
> rm(x.3)
> ls()
```

```
## [1] "x"   "X"
## [3] "x_2" "x1"
```

```
> rm(x1, x_2)
> ls()
```

```
## [1] "x" "X"
```

(`rm(list = ls())` will remove everything)

Basic R Commands: exercises

- Create a new object called 2018data that takes the value 433
- Create the following object (with assigned value):

  `numberDeaths <- 23,968`

- What is y1 equal to?

```
> y1 <- y2 <- 18
```

- What is y3 *finally* equal to?

```
> y3 <- 1
> y3 <- 2
> y3 <- y3 + 5
```

Help Files

Help Files: intro

- Several examples of `R` *functions* have been introduced
  - `dir()`, `getwd()`, `ls()`
- We can learn more from `R`'s help files
  - What arguments does the function accept?
  - Search for a function that meets your needs
  - Examples of how to use a function
  - Related functions that are useful to the target
  - What functions are included in a package?

Help Files: commands

For the following examples, we'll use a few new functions

- There are 2 ways to access the help files of a function
  - `help("mean")` or `?"mean"`
- We can the help files for functions
  - `help.search("median")`
  - `help.search("standard dev")`
- Related functions are grouped together in **packages**
  - `help(package = stats)`

Help Files: contents

- **Description** & **Usage** – just that
- **Arguments** – the inputs needed to produce output(s)
- **Value** – the outputs
- **See Also** – VERY useful (especially if you are searching for a new tool and you don't know the name)
- **Examples** – sometimes this is what I look at first

Help Files: exercises

- Earlier, we learned how to list the files in the current working directory. How do you get `R` to show you .Rhistory?
- Find a function that calculates the variance.
- Find a function that creates a scatterplot.

Data Structures

Data Structures: overview

- A few types of data in `R`
  - `logical` - `TRUE` or `FALSE`
    - `NA` - missing data
  - `character`
  - numeric - `integer`, `double`, `complex`
  - `NULL` - empty object (place holder)
- `R` has different structures for holding data, which can be organized by...
  - how many dimensions does it have?
  - do the types of data need to be the same?

Data Structures: overview (continued)

- **Vectors**
  1. 1 dimension
  2. same data type
  - special case: **factor** (predefined categories)
- **Matrices**
  1. rows and columns
  2. same data type
- **Arrays**
  1. any number of dimensions
  2. same data type
- **Data Frames**
  1. rows and columns
  2. different data types
- **Lists**
  1. any number of dimensions
  2. different data types

Data Structures: vectors

Several ways to create vectors. . .

```
> vector1 <- c(1, 2, 3, 4, 5) # c() for combine elements
> vector1
```

```
## [1] 1 2 3 4 5
```

```
> mean(vector1)
```

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

```
> vector2 <- 1:10
> vector2
```

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

```
> vector3 <- c("a", "b", "c", "54", "zebra",
+              "OSU", "scarlet", "grAY23423")
> mean(vector3)
```

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

Data Structures: exercises with vectors

- Create a new vector called vector4 that is exactly the same as vector3.
- Create a new vector called vector5 that is exactly the same as vector3 but includes the string "IPR" at the end (i.e., as the 9th element).
- What does the function `seq()` do? Use this function in two different ways.
- Make a vector with elements 100, 99, ..., -98, -99, -100. Please, please, please, please do not type in each individual number..

Data Structures: getting to know your vector

A few useful tools when working with vectors

- List info about data structure: `str()`

```
> str(vector1)
```

```
##  num [1:5] 1 2 3 4 5
```

- Number of elements: `length()`

```
> length(vector1)
```

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

- view particular elements in the vector

```
> vector1; vector1[2]
```

```
## [1] 1 2 3 4 5
```

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

Data Structures: `factor` (special kind of vector)

- A `factor` has predefined values for categorical data
  - There are ordered and unordered `factors`
- The function: `factor()` includes arguments...
  - `x` - the actual data
  - `levels` - the predefined values
  - `labels` - informative names for the `levels`

Data Structures: factor (continued)

```
> z <- factor( x = c(0,0,1,1,2),
+              levels = 0:2,
+              labels = c("neonate",
+                         "child",
+                         "adult")
+      )
> is.factor(z)
```

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

```
> z
```

```
## [1] neonate neonate child   child   adult
## Levels: neonate child adult
```

```
> levels(z)
```

```
## [1] "neonate" "child"   "adult"
```

Data Structures: creating matrices

```
> # cbind() - combine vectors as columns
> Mat1 <- cbind( c(1:2), c(3:4), c(5:6) )
> Mat1
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
> # rbind() - combine vectors as rows
> Mat2 <- rbind( c(1:2), c(3:4), c(5:6) )
> Mat2
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

Data Structures: getting to know your matrices

- With vectors, we used `vector1[i]` to access the i^{th} element.
- With matrices, the location of an element has 2 parts: *row* & *column*
- Suppose, `coolMatrix` is a 10 by 10 matrix. We can access the element in the $3^{rd} row and the $8^{th} column as follows...
  - `coolmatrix[3, 8]`
- Excluding the row will return an entire column
  - `coolMatrix[, 8]`
  - or you can get part of a column

    `coolMatrix[c(3:5), 8]`

Data Structures: getting to know your matrices

```
> dim(Mat1)
```

```
## [1] 2 3
```

```
> Mat1[1,1]
```

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

```
> t(Mat1)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

```
> str(Mat2)
```

```
##  int [1:3, 1:2] 1 3 5 2 4 6
```

```
> nrow(Mat2)
```

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

```
> ncol(Mat2)
```

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

```
> Mat2[,2]
```

```
## [1] 2 4 6
```

Data Structures: exercises for vectors & matrices

- How do you print out the first row of Mat2?
- Replace the diagonal elements in Mat1 with zeros.
- Why does the following code give an Warning message?

```
> M1 <- rbind( c("a", "b"), c("d", "e") )
> cbind( M1, c("1", "2", "3") )
```

- Find another `R` function for creating a matrix, and provide an example.
- Create a 2 x 6 matrix called Mat21 that combines Mat1 and t(Mat2).

Data Structures: data frames intro

- **Data frames** are `R` structure for typical data sets (i.e., variables as columns and an observation for each row).
- To explore data frames we are going to use a new function.
  - `read.csv()` – read in a CSV file (the resulting object will be a data frame)
- Load new Project in R Studio Cloud

Data Structures: data frames example

```
> dir()
```

```
## [1] "interva.R"          "introR_data1.csv" "template.R"
```

```
> data <- read.csv("introR_data1.csv")
> is.vector(data)
```

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

```
> is.factor(data)
```

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

```
> is.matrix(data)
```

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

```
> is.data.frame(data)
```

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

Data Structures: data frames example (continued)

A few useful tools to work with data frames...*

```
> str(data)  ## print structure of data frame
```

```
## 'data.frame':    50 obs. of  10 variables:
##  $ X          : Factor w/ 50 levels "Alabama","Alaska",..:
##  $ Population: num  3615 365 2212 2110 21198 ...
##  $ Income     : Factor w/ 49 levels "","3098","3378",..: 6
##  $ Illiteracy: num  2.1 1.5 1.8 -1.9 1.1 0.7 1.1 0.9 1.3 2
##  $ Life.Exp   : num  69 69.3 70.5 70.7 6 ...
##  $ Murder     : num  15.1 11.3 7.8 10.1 10.3 6.8 NA 6.2 10.
##  $ HS.Grad    : num  413 66.7 58.1 39.9 62.6 63.9 56 54.6 5
##  $ Frost      : Factor w/ 37 levels "0","100","101",..: 37
##  $ Area       : int  50708 566432 113417 51945 156361 10376
##  $ Governor   : Factor w/ 12 levels "","D","Dem","Demo",..:
```

Data Structures: data frames example (continued)

A few useful tools to work with data frames. . . *

```
> names(data)  ## list variable names
```

```
##  [1] "X"          "Population" "Income"     "Illiteracy"
##  [5] "Life.Exp"   "Murder"     "HS.Grad"    "Frost"
##  [9] "Area"       "Governor"
```

```
> dim(data)    ## dimensions (rows, columns)
```

```
## [1] 50 10
```

```
> newDFrame <- as.data.frame(Mat1)  ## convert matrix to d.f.
> newDFrame
```

```
##   V1 V2 V3
## 1  1  3  5
## 2  2  4  6
```

Data Structures: data frames example (continued)

A few useful tools to work with data frames...*

```
> data$Life.Exp[1:2]                ## access variable with $
```

```
## [1] 69.05 69.31
```

```
> summary(data$Life.Exp)            ## summarize variable
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    6.00   70.16   70.89 1062.38   72.39 9999.00
```

```
> ## summary(data)                  ## summarize all variables
> data$NewVar <- 1:nrow(data)       ## create new variable
> names(data)
```

```
##  [1] "X"          "Population" "Income"     "Illiteracy"
##  [5] "Life.Exp"   "Murder"     "HS.Grad"    "Frost"
##  [9] "Area"       "Governor"   "NewVar"
```

Data Structures: data frame exercises

- Summarize the Population variable and describe the values.
- Write an R command that prints out the 7th, 12th, and 33rd state names.
- Write an R command that sorts the state names in reverse alphabetical order.
- Create a new variable that contains the state names, but in lower case letters.
  - *hint* use the `apropos()` function to search (or Google)

Data Structures: lists

What are the characteristics of a list in R?

```
> newList <- list(
+    v1 = c("a", "b", "c"),
+    v2 = 1:4
+    )
> newList
```

```
## $v1
## [1] "a" "b" "c"
##
## $v2
## [1] 1 2 3 4
```

```
> newList$v1
```

```
## [1] "a" "b" "c"
```

```
> newList$v2
```

```
## [1] 1 2 3 4
```

```
> newList$v3 <- c(
+ "x", "y", "z"
+ )
> newList
```

```
## $v1
## [1] "a" "b" "c"
##
## $v2
## [1] 1 2 3 4
##
## $v3
## [1] "x" "y" "z"
```

Functions

Functions: intro & example

- While `R` has many useful tools, all your needs may not be covered.
  - *solution* create your own tool!
- Simple example of `function()`

```
> addNum <- function(num1, num2) { # define fnc name & args
+   answer <- num1 + num2          # code to be executed
+   return(answer)                 # what the function
+ }                                #   returns
>
> y <- addNum(3, 4)
> y
```

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

Functions: exercises

- Create a new function that accepts 3 arguments, and returns the sum.
- Create a new function that returns the summary of a variable.

Installing Packages

Installing Packages: basics

- Most R packages can be installed and loaded with ease.

```
> install.packages("CrossVA")
```

- On your own computer, R will ask you...
  - to choose a mirror from where you would like to download the package
  - and specify a folder where you would like to install the package (R will suggest a location)
- After installing the package we can load it (and thus have access to the functions) with as follows

```
> library(CrossVA)
```

Installing Packages: `rJava`

- The `rJava` package allows `R` to call (*usually faster*) Java programs
  - `R` is an extremely flexible programming language, which makes it relatively slow (among other reasons).
  - This is what the `InSilicoVA` package does.
- Configuring `rJava` may require an additional step or two (compared to other packages).
  - (RStudio is usually pretty good at figuring this out on its own.)
  - Mac & Linux: within a terminal...

```
> R CMD javareconf
```

Installing Packages: rJava on Windows

On Windows, try one of the following commands at the R prompt...

```
> ## option 1
> options(java.home = "C:\\Path\\to\\Java\\jdk")
```

```
> ## option 2
> Sys.setenv("JAVA_HOME" = "C:\\Path\\to\\Java\\jdk")
```

If these fail, try adding the environment variable JAVA_HOME set equal to the path to jdk (and restarting your computer).