Introduction to R

Cecina Babich Morrow

R Basics – The Console

You can use R as a calculator:

```r
2+2
```

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

```r
3^6
```

```
## [1] 729
```

```r
(8+2)/5
```

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

Try it yourself: Evaluate $\frac{1+\sqrt{5}}{2}$ using R. Hint: use sqrt() for the square root.

Just like in Python, you'll want to name variables and assign them values. You can either use "=" or "<-" to assign values to variables:

```r
number_interns = 6
your_name <- "Marie Curie" #This isn't really your name
```

Try it yourself: Change the name in your_name to (you guessed it) your own name and add a more useful comment. The "#" designates comments in R, just like in Python.

Notice that when you ran that chunk of code, these variables appeared in the Environment tab of RStudio. We can check the data type for these variables with the class() function:

```r
class(number_interns)
```

```
## [1] "numeric"
```

```r
class(your_name)
```

```
## [1] "character"
```

Try it yourself: We've seen numeric and character data types. Try to make a Boolean variable. Check yourself using the class() function.

Projects

Most of the time, we want to save our work in R as part of a project. In general (don't do this now!), to create a new project, go to File -> New Project -> New Directory -> Empty Project. You can browse to whatever folder you want to create your project.

R + GitHub

For this internship, we want to use GitHub to save our code. Start by going to GitHub in your browser: https://github.com/babichmorrowc/SlothSquad.

Scripts

So far, you've been typing your R commands directly into the console window to run. R scripts, like Python scripts, are a way to write and save a set of R commands. Go to File -> New File -> R Script. Start your R script with some helpful comments, including your name, the date, and the purpose of the script, for example:

```
#Cecina Babich Morrow
#10/4/2018
#Test R script
```

Pick your favorite calculation and type it in the script. Notice what happens when you press Enter. To run that line of code in the console, you can either click anywhere on that line and press the Run button in the top right of the script menu. Alternatively, you can press Command+Enter to run the line.

Let's get started!

Now that we have a test script, we can start messing around with some code. One very common type of data in R is a vector, which is a one-dimensional array that can hold numerical, character, or logical data. To create a vector, use the c() function:

```
intern_names <- c("Marie Curie", "Dorothy Vaughan", "Jane Goodall", "Rosalind Franklin", "Grace Hopper"
```

But wait, these still aren't the right names. Let's start with the first element of the intern_names vector, which is currently "Marie Curie". Unlike in Python, **indices in R start with 1** rather than 0 – this might be a bit of a pain to remember, so just make sure you have the right index before you modify anything.

```
intern_names[1]
```

```
## [1] "Marie Curie"
```

Try it yourself: Change the first element in intern_names to your own name. Hint: you can reassign elements in a vector in the same way you would reassign a variable.

Let's add some numerical data:

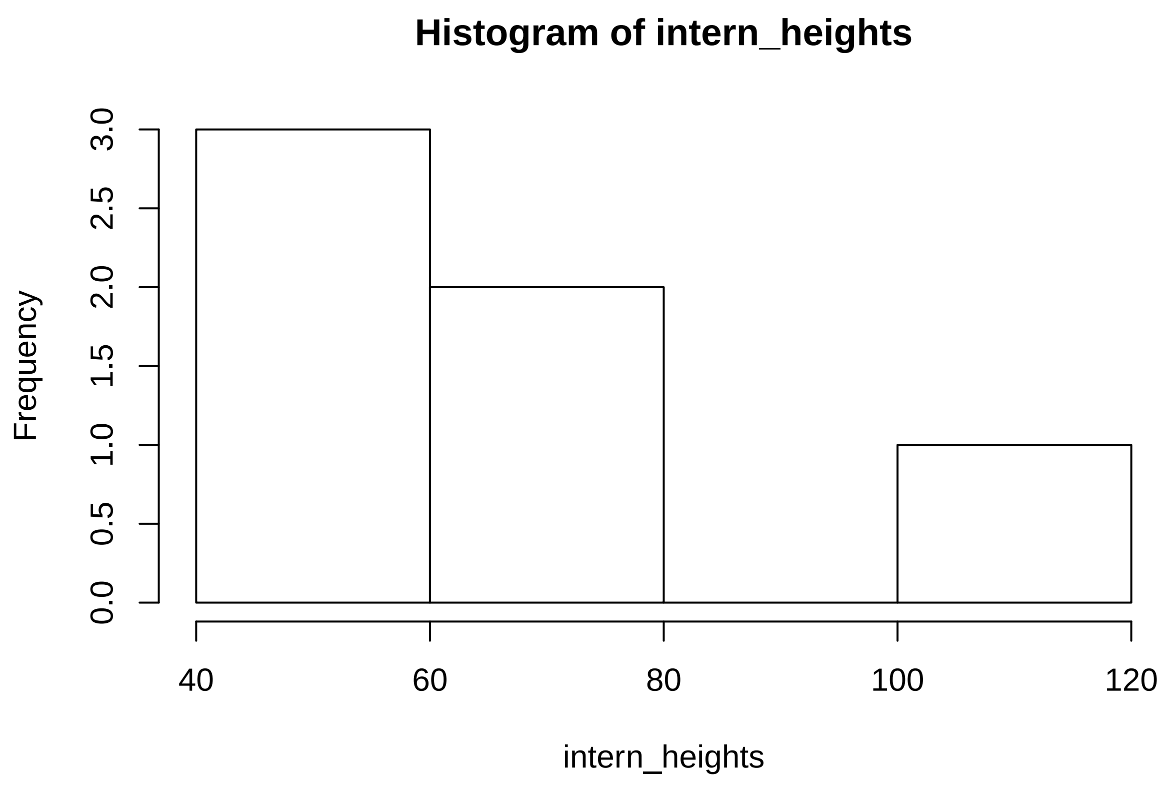
```
intern_heights = c(42, 45, 110, 68, 54, 76)
```

Make sure you enter heights in the same order as the names in your intern_names vector. In order to keep track of the list of numbers you just entered, you can name the intern_heights vector using the intern_names vector you've already created:

```
names(intern_heights) <- intern_names
```

Now that we have some numerical data, we can even make a plot. There are many different ways to make graphs in R (with varying levels of complexity/prettiness), but the base package includes basic plotting functions. We can make a histogram of your heights:

```
hist(intern_heights)
```



Adding more data

Not a very beautiful histogram… we don't have enough data for a really useful visualization.

To get some more data to play with, we're going to use one of R's many packages.

R Packages

Packages are part of what makes R so special: a package is basically a collection of functions, help menus, data, vignettes, etc. stored in a (wait for it) package. You can think of a package kind of like a lightbulb: if you want to start using it, you have to first put it in the lamp (install it). You only have to install an R package once. Everytime you want to use the lightbulb, though, you have to turn it on–everytime you want to use an R package, you must load it:

```
#Install the yarrr package (from https://bookdown.org/ndphillips/YaRrr/):
#you'll have to uncomment the following line the first time to install the package initially
#install.packages("yarrr")
#Load the package:
library(yarrr)
#you could also run require(yarrr) -- there's a subtle difference, but for most uses they are equivalen
```

This package has some data called "pirates" available for us.

Exploring a dataset

Let's take a look at the first few rows of the data. The function head() does basically the same thing it did in Python:

```r
head(pirates)
```

```
##   id    sex age height weight headband college tattoos tchests parrots
## 1  1   male  28 173.11   70.5      yes   JSSFP       9       0       0
## 2  2   male  31 209.25  105.6      yes   JSSFP       9      11       0
## 3  3   male  26 169.95   77.1      yes    CCCC      10      10       1
## 4  4 female  31 144.29   58.5       no   JSSFP       2       0       2
## 5  5 female  41 157.85   58.4      yes   JSSFP       9       6       4
## 6  6   male  26 190.20   85.4      yes    CCCC       7      19       0
##   favorite.pirate sword.type eyepatch sword.time beard.length
## 1    Jack Sparrow    cutlass        1       0.58           16
## 2    Jack Sparrow    cutlass        0       1.11           21
## 3    Jack Sparrow    cutlass        1       1.44           19
## 4    Jack Sparrow   scimitar        1      36.11            2
## 5            Hook    cutlass        1       0.11            0
## 6    Jack Sparrow    cutlass        1       0.59           17
##              fav.pixar grogg
## 1      Monsters, Inc.    11
## 2               WALL-E     9
## 3           Inside Out     7
## 4           Inside Out     9
## 5           Inside Out    14
## 6 Monsters University     7
```

Try it yourself: Run a command that will allow you to see the last 5 rows of the dataset…the last 10?

Now let's try another histogram, this time using the 1000 pirate heights. What happens if we try basically the same thing we did for intern_heights?

```r
hist(pirates)
```

```
## Error in hist.default(pirates): 'x' must be numeric
```

We want a histogram of the height column specifically, so we need to tell R that. The "$" symbol selects a column from a dataset.

```r
head(pirates$height)
```

```
## [1] 173.11 209.25 169.95 144.29 157.85 190.20
```

Try it yourself: Now try creating a histogram for the pirate heights!

One of the most useful parts of RStudio is the Help menu: you can type the name of any function into the search box and read about the documentation. Learning to read a help page can be tricky, but it's a really useful tool.

Try it yourself: Use the help function for hist to customize your histogram of pirate heights. Some ideas: change the title, change the x-axis label, change the number of cells in the histogram, whatever you want!

Make sure to save your script with all the code you've run today!

Recap

Stuff you've done in R!

- Become familiar with the RStudio set-up
  - console, projects, scripts, environment, and help
- Created and modified variables

    - multiple types of variables
    - vectors (Remember: **indexes start with 1**)
- Installed and loaded a package
- Worked with a (fairly large) dataset
    - created a histogram to visualize that data