1: Intro to Basics

Introduction to R

I took the following from the Free Introduction to R tutorial from DataCamp(http://datacamp.com/courses/free-introduction-r). It was free but they started charging for five of the modules sometime before I started my course. This is a one-off in case I can't get free access for my students.

How this works

This assignment consists of six parts:

- Intro to Basics (this document)
- Vectors
- Matrices
- Factors
- Data frames
- Lists

Each part has two files, the instructions (this document) and an R Notebook answer sheet where you type your code. You must have both files.

This document contains a number of small exercises that will help you learn the basics of R. Each exercise contains information about some aspect about R, instructions for you to follow. The R Notebook has code chunks where you will type and run your code. You can run each block by clicking the small green arrow in the upper right of each block. When you click the arrow, every line of code is interpreted and executed by R. If your code is correct, it will appear below the block of code. If the code is incorrect, you will get an error message. If you get an error, fix the code

After you complete each exercise, push the R Notebook to your remote repo. Do *not* push this document.

1.1 Comments and calculators

R makes use of the # sign to add comments, so that you and others can understand what the R code is about. Just like Twitter! Comments are not run as R code, so they will not influence your result. For example, Calculate 3 + 4 in the example below is a comment.

```
# Calculate 3 + 4
```

You can execute R commands in your R Notebook but you can also execute R commands straight in the console of RStudio. This is a good way to experiment with R code, to be sure you have it right before you finalize your answers.

Instructions

Here is some sample code. Can you see which lines are actual R code and which are comments?

Add a line of code in the space provided that calculates the sum of 6 and 12, and press the green arrow to run your code chunk.

Note: Get in the habit of commenting your code. It is better to use too many comments that not enough. You will return to your code at some point in the future. The comments will help you remember what you did and *why* you did it.

```r
# Calculate 3 + 4
3 + 4

# Calculate 6 + 12



#
```

1.2 Arithmetic with R

In its most basic form, R can be used as a simple calculator. Consider the following arithmetic operators:

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`
- Modulo: `%%`

The last two might need some explaining:

- The `^` operator raises the number to its left to the power of the number to its right: for example `3^2` is 9.
- The modulo returns the remainder of the division of the number to the left by the number on its right, for example 5 modulo 3 or `5 %% 3` is 2.

With this knowledge, follow the instructions below to complete this exercise.

Instructions

- Type `2 ^ 5` in the editor to calculate 2 to the power 5.
- Type `28 %% 6` to calculate 28 modulo 6.

Click the green arrow and have a look at the R output below the code. Note how the # symbol is used to add comments on the R code but is not included in the outout.

```r
# An addition
5 + 5

# A subtraction
5 - 5

# A multiplication
3 * 5

 # A division
(5 + 5) / 2

# Exponentiation



# Modulo
```

```
#
```

1.3 Variable assignment

A basic concept in (statistical) programming is called a **variable.**

A variable allows you to store a value (e.g., 4) or an object (e.g., a function description) in R. You can then later use this variable's name to easily access the value or the object that is stored within this variable.

You can assign a value 4 to a variable `my_var` with the command. In RStudio, press `option/alt + -` to enter the `<-` with appropriate spacing. Or, take the long way and type the less than symbol followed by a dash. Your choice. *Work smart, not hard.*

You can see what is stored in a variable by entering the variable name on a line by itself.

```
my_var <- 4

my_var
```

Instructions

Complete the code in the chunk such that it assigns the value 42 to the variable `x`, and then view the contents of `x`. You know what to do with the green arrow. Notice that when you ask R to print `x`, the value 42 appears. At least it will if you did this correctly!

```
# Assign the value 42 to x

# Print out the value of the variable x


#
```

1.4 Variable assignment (2)

Suppose you have a fruit basket with five apples. As a data analyst in training, you want to store the number of apples in a variable with the name `my_apples`.

Instructions

- Type the following code in the code chunk: `my_apples <- 5`. This will assign the value 5 to `my_apples`.
- Type `my_apples` below the second comment. This will print out the value of `my_apples`.

Run the code chunk and look at the console: you see that the number 5 is printed. So R now links the variable `my_apples` to the value 5.

```
# Assign the value 5 to the variable my_apples


# Print out the value of the variable my_apples


#
```

1.5 Variable assignment (3)

Every tasty fruit basket needs oranges, so you decide to add six oranges. As a data analyst, your reflex is to immediately create the variable `my_oranges` and assign the value 6 to it. Next, you want to calculate how many pieces of fruit you have in total. Since you have given meaningful names to these values, you can now code this in a clear way:

```
my_apples + my_oranges
```

Instructions

- Assign to `my_oranges` the value 6.
- Add the variables `my_apples` and `my_oranges` and have R simply print the result.
- Assign the result of adding `my_apples` and `my_oranges` to a new variable `my_fruit`.

```
# Assign a value to the variables my_apples and my_oranges
my_apples <- 5


# Add these two variables together


# Create the variable my_fruit


#
```

1.6 Apples and oranges

Common knowledge tells you not to add apples and oranges. But hey, that is what you just did, no? The `my_apples` and `my_oranges` variables in the previous exercise both contained a number. The `+` operator works with numeric variables in R. If you really tried to add "apples" and "oranges", and assigned a text value to the variable `my_oranges`, you would be trying to assign the addition of a numeric and a character variable to the variable `my_fruit`. This is not possible.

```
# Assign a value to the variable my_apples
my_apples <- 5

# Fix the assignment of my_oranges
my_oranges <- "six"

# Create the variable my_fruit and print it out
my_fruit <- my_apples + my_oranges
```

Instrutions

- Run the code chunk and read the error message. Make sure to understand why this did not work.
- Adjust the code so that R knows you have 6 oranges and thus a fruit basket with 11 pieces of fruit.

```
# Assign a value to the variable my_apples
my_apples <- 5

# Fix the assignment of my_oranges
```

```r
my_oranges <- "six"

# Create the variable my_fruit and print it out
my_fruit <- my_apples + my_oranges

my_fruit
```

1.7 Basic data types in R

R works with numerous data types. Some of the most basic types to get started are:

- Decimal values like 4.5 are called **numerics**.
- Natural numbers like 4 are called **integers**. Integers are also numerics.
- Boolean values (TRUE or FALSE) are called **logical**.
- Text (or string) values are called **characters**.
- Note how the quotation marks around the text in the code chunk indicate that `"some text"` is a character.

Instructions

Assign values to the these variables:

- `my_numeric` variable to 42.
- `my_character` variable to "universe". Note that the quotation marks indicate that "universe" is a character.
- `my_logical` variable to `FALSE`. Note that R is case sensitive!

```r
# Assign 42 to my_numeric
my_numeric

# Assign "universe" to my_character
my_character

# Assign FALSE to my_logical. Remember: Case matters
my_logical
```

1.8 What is that data type?

Do you remember that when you added `5 + "six"`, you got an error due to a mismatch in data types? You can avoid such embarrassing situations by checking the data type of a variable beforehand. You can do this with the `class()` function, as the code on the right shows.

Instructions

- Complete the code to show the classes of `my_character` and `my_logical`.

```r
# Declare variables of different types
my_numeric <- 42
my_character <- "universe"
my_logical <- FALSE
```

```r
# Check class of my_numeric
class(my_numeric)

# Check class of my_character


# Check class of my_logical


#
```