Part 1: Intro to Basics

Introduction to R

This assignment consists of six parts:

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

Each part introduces you to a fundamental type of *object* that R uses to store data. You will also learn basic R commands necessary to store and access data in the objects.

Later, you will create an R script, a text file that stores your R commands so that you don't have to rewrite code. You will write scripts for most assignments.

After you complete each exercise, push the R script to your remote repo. See Part 0 for instructions. Do *not* push this document.

1.1 Comments and calculators

Open R Studio and click in the **Console** panel. It's probably the lower left pane in R Studio. Be sure the Console tab is chosen.

You should see

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

followed by the `>` (greater than) prompt. The prompt is where you can type commands directly into R.

Click just to the right of the `>`, type `42`, and press the Enter key. You should see `[1] 42`. That is R's way of responding to your command. You can use R entirely from the console but it's not efficient. Instead, you will write scripts. Scripts are plain text files with all of the commands needed to accomplish one or more related tasks. You can then run (execute) all or some of the commands in your script.

Writing your first script

Click in the `hw01-1.R` script you created. It should be the upper-left panel in RStudio.

Type a `#` followed by your name. R uses # sign to add comments, so that you and others can understand what the R code is about. Add another comment line followed by this assignment number and name. It should look similar to this.

```
# Mike Taylor

# HW02 Part 1: Intro to Basics

# You can add other comments if you want.
```

Comments are not run as R code, so they will not influence your results. For example, the code below shows a comment ("Calculate 2 + 5"), followed by the actual code, then another comment. Only the code is executed by R.

```
# Calculate 2 + 5

2 + 5

# The answer should be 7
```

You execute R commands in your R scripts (and later, R Notebooks) in one of two ways. To run a single line, click at the end of line withe the command you want to run (or anywhere in the line) and press the Cmd/Alt-enter (enter or return key). You can also press the small `Run` button just above your script, to the right. You'll see the results appear in the console below the script.

You can select several lines of related code and then do the same thing to execute all of the lines at once.

Instructions

Add a line of new line code in your script that calculates 30 + 12. Run the line of code. Write a comment above your new code that describes what your code does.

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

You would not normally write a comment for every step. The code to add 30 + 12 is obvious. For now, write comments for most steps to develop good coding habits.

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 `^` (caret) operator raises the number to its left to the power of the number to its right: for example `3^2` is 3 squared, or 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. Five divided by three has a remainder of 2.

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

Instructions

- Enter the comments shown below.
- For Division, enter `121/1.1` to calculate 121 divided by 1.1.
- For Exponentiation type `2^5` in your script to calculate 2 to the power 5.
- For Modulo, type `28 %% 6` to calculate 28 modulo 6.

Run the commands to be sure you get the correct results of 110, 32, and 4.

```
# Section 1.2

# Division


# Exponentiation



# Modulo



#
```

R follows the mathematical order of operators, or precedence. Add these two lines of code to your script to see the result of exponentiation without and with parentheses.

- `49^1/2`
- `49^(1/2)`

```
# Exponentiation without parentheses


# Exponentiation with parentheses


#
```

The results of 24.5 and 7 are very different. *Failure to remember precedence will cause mistakes in your results!*

You can do these simple math problems more quickly on a calculator than it takes to start RStudio and enter the commands. The next steps begin to reveal the real power of programming with R.

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 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 `my_var <- 4`. 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. Learn the keyboard shortcuts!*

You can see what is stored in a variable by entering the variable name on a line by itself. You can enter `my_var` in the console or run it as part of your script.

```
my_var <- 4

my_var
```

Note: The `<-` symbol is the *assignment operator*. It tells R to store something (like a value) in something else (like a variable). A handy name for `<-` is "gets." For example, read the command `my_var <- 4` as "my_var gets 4."

Instructions

- Add to your script a comment with the number and name of this section ("1.3 Variable assignment").
- Add to your script the commands that assigns the value 42 to the variable `x`, and then shows the contents of `x`. Notice that when you ask R to print `x`, the value 42 appears in the console. At least it will if you did this correctly!

```
# Assign the value 42 to x


# Print out the value of the variable x


#
```

Let's get biological

Assume you surveyd the number and species of Kangaroo Rats (genus *Dipodomys*) from a transect in southern New Mexico. You recorded 24 *Dipodomys merriami* and 9 *Dipodomys ordii*.

Instructions

- Enter the code to assign the number of *D. merriami* to a variable called `d_merriami`. Assign the number of *D. ordii* recorded to a variable of a similar, suitable name.
- Enter the two variable names on separate lines in your script so you can inspect the results.

Run your code and look at the console for the results. You should see each variable name followed by the number of individuals observed (24 *D. merriami* and 9 *D. ordii*.) If the output does not agree, check your code and fix it so that it runs correctly.

```
# Assign the value 24 the variable d_merriami


# Assign the value 9 to d_ordii


# Check the results
```

1.4 Working with variables

Enter a comment with the section number and name of this section shown above. Notice how you are using comment lines to break your code up into logical chunks of related commands. Use multiple `#` on a line to create visual breaks in your code, like

```
### 1.4 Working with variables
```

You can change the values stored in variables. Let's assume the 24 *D. merriami* should have been 29 so you want to add 5 more individuals. Read the following code and predict the result. What will be the final value of `d_merriami`? Enter the code into your script and run it.

```
# Correcting the mistake
d_merriami + 5

d_merriami
```

```
## [1] 29
```

```
## [1] 24
```

The value of `d_merriami` is still 24, even though R output 29 as the result of `d_merriami + 5`. You must use the assignment operator to store your new value. This time, R adds 5 to the initial value stored in `d_merriami` and then assigns the new value to the variable.

```
# Correcting the mistake
d_merriami <- d_merriami + 5

d_merriami
```

```
## [1] 29
```

You can apply math operators to variables with numbers. For example, `z` holds the value `42` after the following code is run.

```
x <- 30
y <- 12

z <- x + y

z
```

```
## [1] 42
```

Instructions

You will calculate the total number of kangaroo rats observed and then the density of individuals per square meter. Assume that you counted the number of rats in 1000 square meters.

Be sure you added the extra five *D. merriami* (29 total) from the example above.

Add the following code to your script.

- Create a variable called `total_k_rats` and add together the number of each species observed. *Use the variable names for addition, not the numbers.*
- Create a variable called `area_sampled` and assign the value `1000` to it.
- Create a variable called `density`. Divide `total_k_rats` by `area_sampled` to calculate the density of kangaroo rats per square meter.

```
# Add together the number of individuals of each species.
# Store the result in 'total_k_rats'.


# Assign 1000 to a variable called 'area_sampled'
```

```
# Divide 'total_k_rats' by 'area_sampled' to calculate 'density'.


# Show the value stored in density


#
```

The correct result should be 0.038 kangaroo rats per square meter. If you did not get that result, the most likely mistake is that you didn't properly add the five extra *D. merriami*.

1.5 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**. Note that these are upper case. R is case sensitive so TRUE is a boolean value but True is not.
- Text (or string) values are called **characters**. Text values in code are surrounded by single or double quotation marks, like `"my text string"`.

Instructions

Assign values to the these variables:

- 42 to the variable `numeric_var`.
- "Dipodomys merriami" to the variable `species_name`. You can use single or double quotation marks but you cannot mix them on the same text string; e.g., `'Dipodomys merriami'` or `"Dipodomys merriami"` but not `"Dipodomys merriami'`.
- `TRUE` to the variable `is_mammal`. Remember that R is case sensitive. Boolean values are not text strings so do not use quotation marks.

```
# Assign 42 to numeric_var


# Assign "Dipodomys merriami" to species_name


# Assign TRUE to is_mammal Remember: Case matters
```

1.6 What is the data type?

Do you think you can add (or substract, multiply, etc.) variables of different data types? What if tried to run the code `5 + "five"`? Try it in the console to see the result.

The error message tells you that you are using a non-numeric argument (`"five"`) witt a binary operator (`+`). Math operators like the plus sign work only with numerics. You cannot mix most data types.

You can use the `class()` function to learn the data type of a variable. For example, enter `class(numeric_var)` to learn the data type of the `numeric_var` variable.

Instructions

- By now, you should be in the habit of using one or more hashtags to create a new section in your code. Be sure you have done so throughout.
- Use the `class()` function to determine the class of the three variables you just created above. Add the code to your script.

```
# Check class of numeric_var


# Check class of species_name


# Check class of is_mammal


#
```