R Handout: Basics

PSYC250

This guide has been written for Windows users and may not apply to Mac users. Sorry!

Install R and RStudio

(credit: https://courses.edx.org/courses/UTAustinX/UT.7.01x/3T2014/56c5437b88fa43cf828bff5371c6a924)

Windows

(A) Install R

1. Open an internet browser and go to www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for Windows" link at the top of the page.
5. Click on the "install R for the first time" link at the top of the page.
6. Click "Download R for Windows" and save the executable file somewhere on your computer. Run the .exe file and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

(B) Install RStudio

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Windows version, and save the executable file. Run the .exe file and follow the installation instructions

Mac

(A) Install R

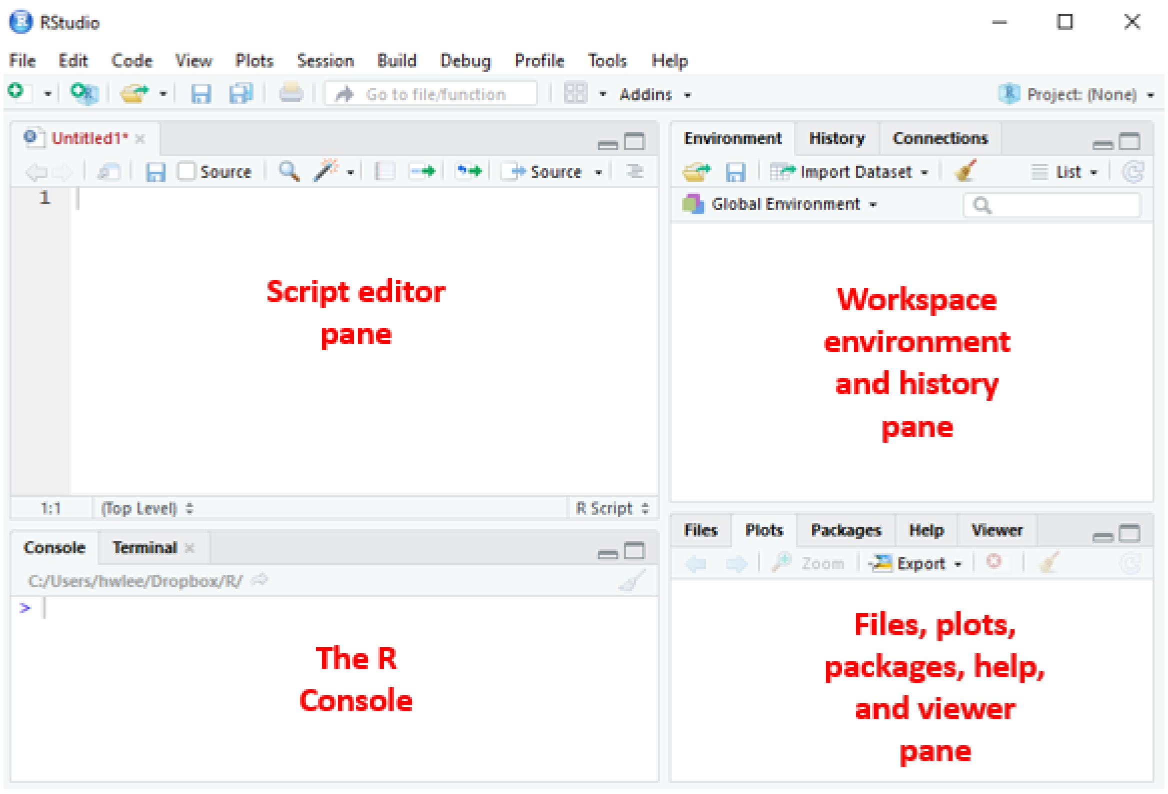
1. Open an internet browser and go to www.r-project.org.
2. Click the "download R" link in the middle of the page under "Getting Started."
3. Select a CRAN location (a mirror site) and click the corresponding link.
4. Click on the "Download R for (Mac) OS X" link at the top of the page.
5. Click on the file containing the latest version of R under "Files."
6. Save the .pkg file, double-click it to open, and follow the installation instructions.
7. Now that R is installed, you need to download and install RStudio.

(B) Install RStudio

1. Go to www.rstudio.com and click on the "Download RStudio" button.
2. Click on "Download RStudio Desktop."
3. Click on the version recommended for your system, or the latest Mac version, save the .dmg file on your computer, double-click it to open, and then drag and drop it to your applications folder.

Navigating RStudio

When we open RStudio, we will see a screen like this:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1*
Source
Source
1:1
(Top Level)
R Script
Script editor pane
Environment History Connections
Import Dataset
List
Global Environment
Workspace environment and history pane
Console Terminal
C:/Users/hwlee/Dropbox/R/
The R Console
Files Plots Packages Help Viewer
Zoom
Export
Files, plots, packages, help, and viewer pane

Note. Because the Terminal tab, the Connections tab, and the Viewer tab will not be used in this course, I will not talk about them.

There are four panes, the script editor pane, the R console pane, the workspace environment and history pane, and the files, plots, packages, help, and viewer pane. Each pane serves different purposes.

1. Upper left pane (**Script editor**): To get R to do stuff (e.g., conduct analyses), we submit commands to R. R then executes those commands. Typically, we type those commands into the script editor and then send the commands from the script editor to the console (see the next point). Although we can type the commands directly into the console, R users prefer to type the commands into the script editor because we can save the commands in the script editor into script files (with the extension .R). The script files allow us to keep long-term records of the analyses that we have conducted. We can also share the script files with other R users so that they can replicate our analyses.
2. Lower left pane (**R console**): The R console is where commands are submitted to R. It is also where we find some of the output from R (e.g., analysis results).
3. Upper right pane (**Workspace**): The environment and history tabs make up the workspace environment. We can also call this R's memory.
   - Under the **Environment** tab, we can find the list of objects (e.g., variables) that we created in the session.
   - Under the **History** tab, we can find all the previous commands we submitted to R.
4. Lower right pane (**Files, plots, packages, help, and viewer**)
   - Under the **Files** tab, we can create new folders on our computer, move, delete, and rename files.

- Under the **Plots** tab, we can find all the plots we instructed R to produce during the session.
- Under the **Packages** tab, we can find, install, and update packages. Packages contain functions that other people have created to supplement those in R.
- Under the **Help** tab, we can find information about a given command or package. We can also find more information about various commands and the packages on this website: https://www.rdocumentation.org/

Some basic commands

It might be useful to know some of these basic commands in R.

1. Place a comment in the script file: Begin the line with #
2. Run a line in the script file: Place the cursor on the line, hit **Ctrl** and **Enter**
    - If we typed the command directly into the console, we only need to hit **Enter**
3. Run multiple lines in the script file: Select the lines, hit **Ctrl** and **Enter**
4. Clear the console: Hit **Ctrl** and **L**
5. Request help: Type a question mark in front of the command or the package name (e.g., **?cor**). Information about the command will appear in the **Help** tab (lower right pane).

Copy and paste the following R codes into the Script Editor pane. Run them to see what they do. Experiment and have fun!

```
# Assign a value (e.g., 9) to an object, say x.
x <- 9

# Get the value for x.
x

# R is case-sensitive. If you typed X, you'll get an error message.
X

# If you want to know what objects are in the workspace (i.e., R's
memory), look at the Environments tab or type ls().
ls()

# You may remove an object (e.g., x) from R’s memory.
rm(x)  # where rm stands for remove

# You may remove all objects from the workspace.
rm(list = ls())

# Assign a non-numerical value by putting the value in quotation marks.
y <- "hello"

# Get value of y. Notice value of y is in quotation marks, indicating it
is a non-numerical value.
y

# Perform mathematical operations in R.
11 + 10
11 - 10
11 * 10
11 / 10
11 ^ 10
11 ^ (1/2)
sqrt(11)  # this number should be the same as above line
```

```
log(11)  # taking natural log (log base e)
log10(11)# taking log base 10
exp(11)  # taking the exponential

# Perform mathematical operations in R with an object (e.g., x).
x <- 9  # we removed x just now, so now, we need to reassign a value to x
x + 10
x - 10
x * 10
x / 10
x ^ (1/2)
sqrt(x)
log(x)
log10(x)
exp(x)

# Perform mathematical operations with more than one object.
y <- 2  # notice that 2 now replaces the value “hello”
x + y
x - y
x * y
x / y
x ^ (1/y)
```

Keying in data directly into R

Before we can conduct any analysis, we need to give R some data. One way is to key the data directly into R.

Let’s say we have three people, Bob, Andrea, and Calvin. We have their ages, the number of children each of them has, and their gender.

Name	Age	No. of Children	Gender
Bob	48	1	Male
Andrea	47	3	Female
Calvin	49	2	Male

First, we create a variable we call *name*, with the three people’s names.

```
# R code for creating string variables (i.e., non-numerical numbers)
name <- c("Bob", "Andrea", "Calvin")  # notice the names are between
quotation marks, which tells R that name is a string (non-numerical)
variable.
```

If we type **name** into the console and hit **Enter**, we get the following output:

```
[1] "Bob"    "Andrea"   "Calvin"
```

Next, we create the variable *age*, with the three people’s ages.

```
# R code for creating numeric variables.
age <- c(48, 47, 49)  # notice the order of age should match the order of
the names
```

Again, we type **age** into the console. We get:

```
[1] 48 47 49
```

Now, we combine the two variables into a data frame, such that each row represents data for one person, and the variables are in side-by-side columns. We use **data.frame()**:

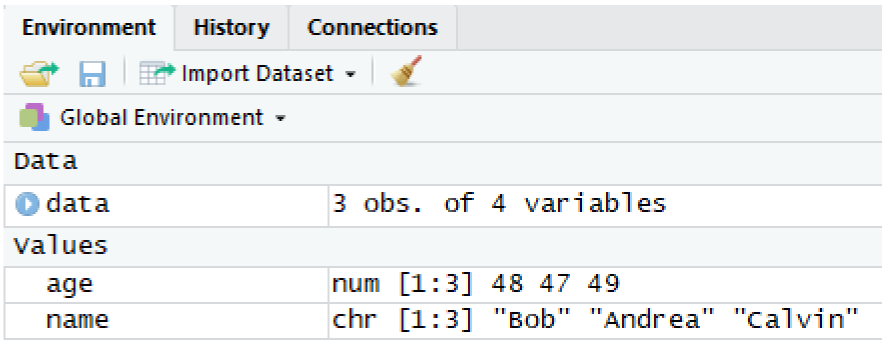
```
# R code for combining variables into a data frame.
# Let's give the data frame a name: data.
data <- data.frame(Name = name, Age = age)
```

We type **data** into the console and get:

```
    Name Age
1    Bob  48
2 Andrea  47
3 Calvin  49
```

The output has two columns. *Name* lists the names of the three individuals. *Age* lists the ages of the three individuals.

The values of *Name* and *Age* in the data frame are copied from the original variables. This means that the original variables, *name* and *age*, are still in R's memory. You can see that this is the case from the **Environment** tab or when you use the **ls()** function.

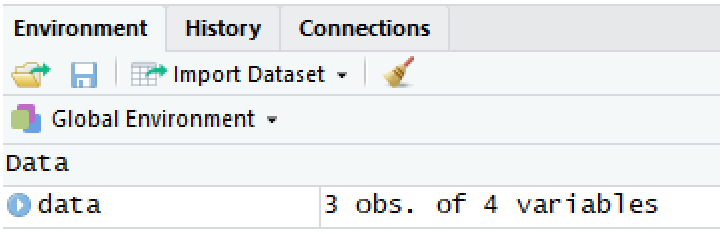


```
Data
data    3 obs. of 4 variables
Values
age     num [1:3] 48 47 49
name    chr [1:3] "Bob" "Andrea" "Calvin"
```

Let's remove the original variables *name* and *age*.

```
# R code for removing variables.
rm(name)
rm(age)
```

Now if you look at the Environment tab, you will see that all that is left is the data frame, *data*.



```
Data
data    3 obs. of 4 variables
```

If we want to use the variables in the data frame *data*, we need to attach **data$** (a dollar sign after the name of the data frame) before the variable name. For example, if we want to know the ages of the three participants:

```
# R code for calling variables in the data frame.
data$Age   # R is case-sensitive so it would not recognize data$age
```

The output will be:

```
[1] 48 47 49
```

Let’s say we have more variables to add to the data frame *data*: the number of children the person has and the person’s gender. Let’s label the number of children each person has as *Children* and the gender of each person as *Gender*.

```
# R code for adding variables into a data frame.
# Let’s label the number of children as Children.
# Let’s label gender as Gender.
data$Children <- c(1, 3, 2)
data$Gender <- c(1, 0, 1)
```

Now, when we type **data** into the console, we get:

```
    Name Age Children Gender
1    Bob  48        1      1
2 Andrea  47        3      0
3 Calvin  49        2      1
```

Note that we created *Children* and *Gender* instead the data frame. So if you type **Children** and **Gender** without **data$**, you will get a warning in the console:

```
> Children
Error: object 'Children' not found
> Gender
Error: object 'Gender' not found
```

At this moment though, we don’t know what 1 and 0 for *Gender* stand for. When we type **data$Gender** into the console, we get this output:

```
[1] 1 0 1
```

There is no indication of what 1 and what 0 refer to. So, we use the **factor()** command to add the levels of the factor.

```
# R code for adding levels to the factor.
data$Gender <- factor(data$Gender, levels = c(1, 0), labels = c("male",
"female"))  # level 1 = label “male”, level 0 = label “female”. Also
notice that male and female are enclosed in quotation marks because these
are string variables.
```

Now, when we type **data$Gender** into the console, we get:

```
[1] male   female male
Levels: male female
```

The first row tells us the first person is male, the second person is female, the third person is male. There is an additional row that tell us that there are two levels of *Gender*: *male* and *female*.

When we type **data** into the console, we get:

```
    Name Age Children Gender
1    Bob  48        1   male
2 Andrea  47        3 female
3 Calvin  49        2   male
```

Instead of 1 and 0, we now have *male* and *female*.

Now, let’s say we made a mistake and need to remove *Children* from the data frame,

```
# R code for removing a variable from the data frame.
data$Children <- NULL
```

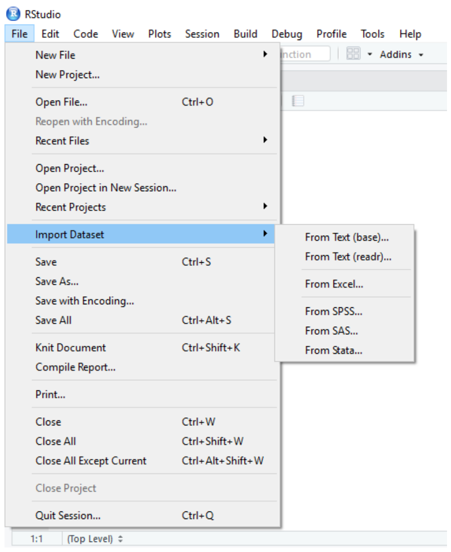
Finally, if we want to save the data frame into a .csv file, we use **write.csv()**.

```
# R code for saving data frame (in .csv format)
write.csv(data, "C:/Users/hwlee/Desktop/R Tutorial/saved.csv")  # specify
the file path (C:/Users/hwlee/Desktop/R Tutorial/) and the name and
extension of the file (saved.csv). The .csv file will now be saved as
saved.csv.
```

Import dataset

Typically, R users do not have to key their data into R directly. Instead, they already have their dataset in various formats, such as .csv, .txt, .sav, .xlsx, etc. So all they need to do is to import the dataset into R. There are several ways to import a dataset.

The most convenient way in RStudio is to go to **File** > **Import dataset** > **(select the format that your file is in)**.



Another convenient way is to use the **file.choose** command. The R codes below show how we would import a .csv file and how we would import a .txt file.

```
# R code to import .csv file: file.choose method.
# Let's give this dataset a name, data.
data <- read.csv(file.choose(), header = TRUE)  # header=TRUE tells R that
the first row has headers.

# R code to import .txt file: file.choose method.
# Let's give this dataset a name, data.
data <- read.delim(file.choose(), header = TRUE)  # header=TRUE tells R
that the first row has headers.
```

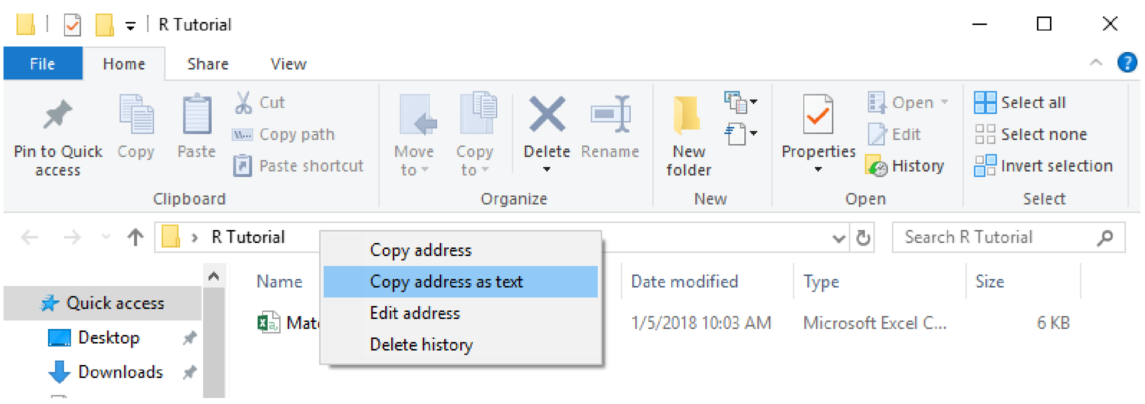
In both cases, a Windows Explorer dialog box will appear, which allows us to go to the folder where the file resides to select it.

Although convenient, I would not recommend the above methods. Researchers tend to have many versions of the same dataset. If there is no record, it might be difficult to remember which dataset was used to run the analyses. That might spell trouble if we need to re-run any analysis.

The way to get around the problem is to specify the file location including the file name (i.e., file path) in the R code. This way, there is a record of exactly which file was used.

```
# R code to import .csv file.
# Let's give this dataset a name, data.
data <- read.table(file = "C:/Users/hwlee/Desktop/R
Tutorial/Materialism.csv", header = TRUE, sep = ",")  # header = TRUE
tells R that the first row has headers, sep = "," tells R that the data
are separated by commas (which is how .csv files separate the data).
```

The italicized part of the R code is the file path. To get the file path, we first go to the folder where the file is located, right-click on the address bar, and click **Copy address as text**.



Let us say the address is C:\Users\hwlee\Desktop\R Tutorial. We add the file name ("Materialism") and its extension (".csv") to the end of this address. Therefore, the file path is C:\Users\hwlee\Desktop\R Tutorial\Materialism.csv.

For the R code to work properly, however, we need to convert the backslashes in the file path to forward slashes. Therefore, we change the file location from C:\Users\hwlee\Desktop\R Tutorial\Materialism.csv to C:/Users/hwlee/Desktop/R Tutorial/Materialism.csv.

If our data are stored in .xlsx format, save the data as .csv format. Then we can use the above method.

If our data are stored in .txt format, then use:

```
# R code to import .txt file.
# Let's give this dataset a name, data.
data <- read.table(file="C:/Users/hwlee/Desktop/R
Tutorial/Materialism.txt", header = TRUE, sep = "\t")  # header = TRUE
tells R that the first row has headers, sep = "\t" tells R that the data
are separated by tabs.
```

To read SPSS data (file extension .sav), we need to install the package **foreign**.

```
# R code to install packages
install.packages("foreign")  # don't forget the ""
# R code to load package
library(foreign) # you must load the package every session you want to use
the package
```

Once we've installed and loaded the package,

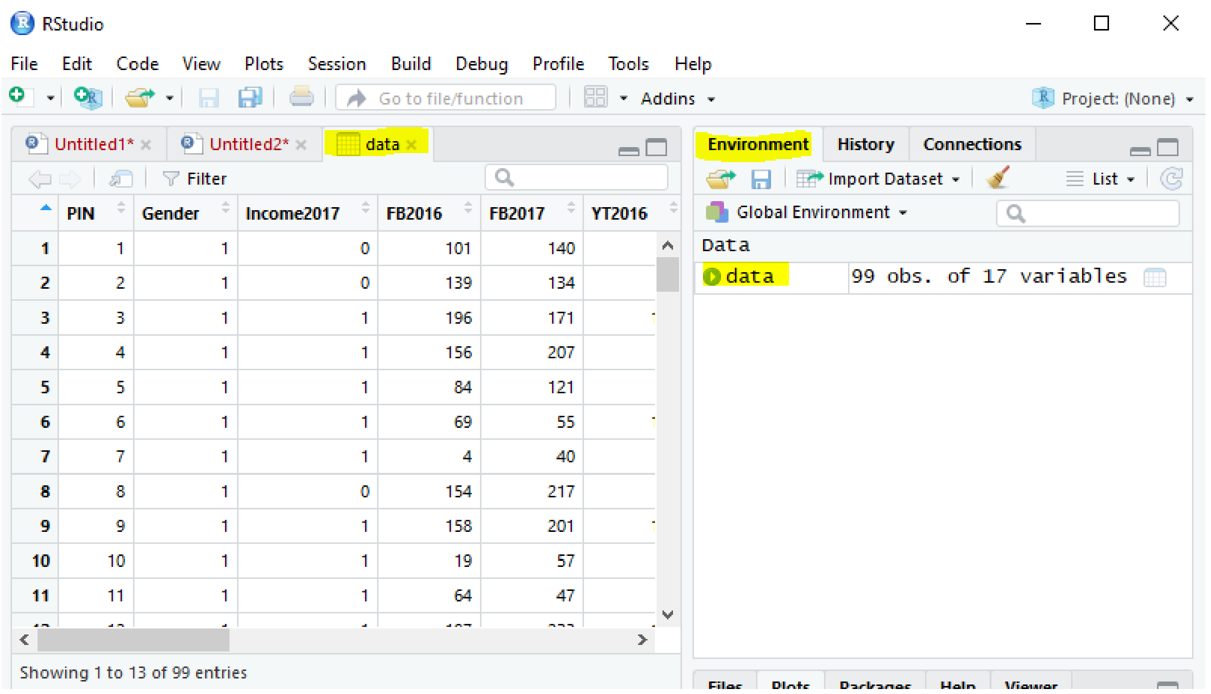
```
# R code to import .sav file
# Let's give this dataset a name, data.
data <- read.spss("C:/Users/hwlee/Desktop/R Tutorial/Materialism.sav",
use.value.labels = TRUE, to.data.frame = TRUE)
```

It may be particularly useful to import .sav files because the factors get imported also. Then we don't have to create the factors. If we imported .csv or .txt file, the factors get imported as integers and we need to convert the integers into factors (See above section on Keying in data directly into R.)

Note. Another package that has similar functions, **haven**, is getting increasingly popular. It allows us to import other formats as well, such as .sas files. If **foreign** does not work for you, you might want to check **haven**.

Check imported dataset

Before conducting any analyses, check that the dataset has been imported correctly. Go to the **Environment pane** (top right pane). Click **data**. The top left pane should show the imported data.



	PIN	Gender	Income2017	FB2016	FB2017	YT2016
1	1	1	0	101	140	
2	2	1	0	139	134	
3	3	1	1	196	171	
4	4	1	1	156	207	
5	5	1	1	84	121	
6	6	1	1	69	55	
7	7	1	1	4	40	
8	8	1	0	154	217	
9	9	1	1	158	201	
10	10	1	1	19	57	
11	11	1	1	64	47	

Showing 1 to 13 of 99 entries

```
Data
data    99 obs. of 17 variables
```

Rows: The data for each participant is recorded in a single row (e.g., data for Participant 1 is in Row 1)
Columns: The data for each variable is recorded in a single column. Names of the variables are in the headers for each column

Scroll down to ensure all rows have been imported correctly. There should be 99 rows. Scroll right to ensure all columns have been imported correctly. There should be 17 columns.

Writing your own functions (optional)

R comes with a lot of functions. If we cannot find a function for our purpose, we might want to install packages, like the **foreign** package. Even then, sometimes, we might not be able to find the correct function. In that case, we might want to write our own functions.

The structure of a function is as follows:

```
# Structure of a function.
function_name <- function(x, y, z) { # arguments/input (x, y, z) the
function takes
  newVar <- x + y                    # do something
  newVar / z                         # return value
}

# Calling (using) a function:
function_name(1, 2, 3)

# Output in the console.
[1] 1
```

In this course, we will focus mostly on the functions already available in base R or in packages. If you'd like to learn more about functions, please visit the following sites for starters:

https://nicercode.github.io/guides/functions/
https://swcarpentry.github.io/r-novice-inflammation/02-func-R/

References

R documentation: https://www.rdocumentation.org/
Mike Marin's YouTube R lecture series:
https://www.youtube.com/watch?v=riONFzJdXcs&list=PLqzoL9-eJTNBDdKgJgJzaQcY6OXmsXAHU
List of useful websites: https://newonlinecourses.science.psu.edu/stat484/node/106/
Essential R: https://newonlinecourses.science.psu.edu/stat484/node/157/
Google's R Style Guide: https://google.github.io/styleguide/Rguide.xml
R Tutor: http://www.r-tutor.com/elementary-statistics