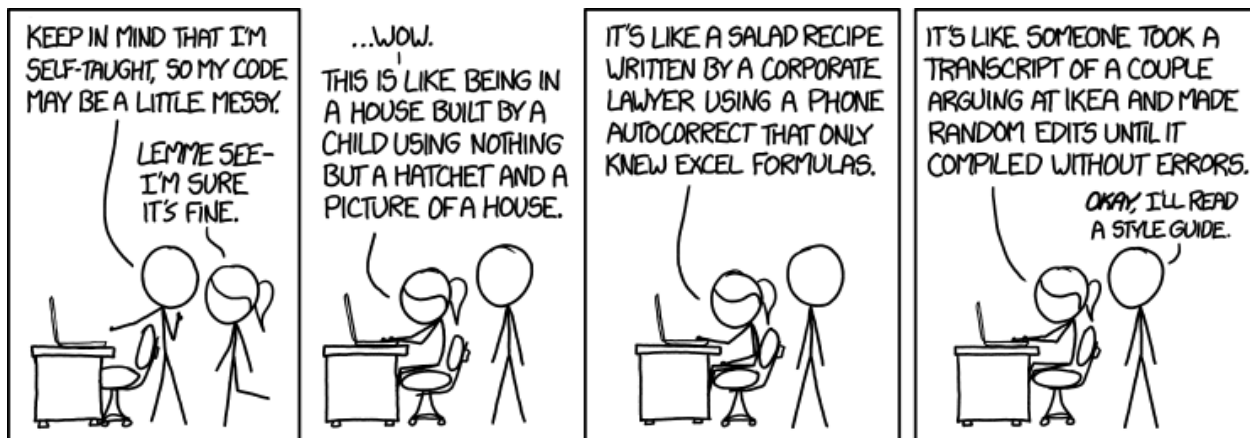


# Tools: R and Rmarkdown

R (<https://www.r-project.org>) is a free, open source software environment for statistical computing and graphics that is available for every major platform.

RStudio (<https://rstudio.com>) is an integrated development environment (IDE) for R. It is also free, open source, and available for every major platform. It makes data analysis and projects in R go a bit smoother.



<https://xkcd.com/1513/>

**Alternative Text:** I honestly didn't think you could even USE emoji in variable names. Or that there were so many different crying ones.

# 1 Getting Started with R

We can use R like an overgrown calculator.

```
# simple math  
5*(10 - 4) + 44
```

```
## [1] 74
```

```
# integer division  
7 %/% 2
```

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

```
# modulo operator (Remainder)  
7 %% 2
```

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

```
# powers  
1.5^3
```

```
## [1] 3.375
```

We can use mathematical functions.

```
# exponentiation  
exp(1)
```

```
## [1] 2.718282
```

```
# logarithms  
log(100)
```

```
## [1] 4.60517
```

```
log(100, base = 10)
```

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

```
# trigonometric functions  
sin(pi/2)
```

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

```
cos(pi)
```

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

```
asin(1)
```

```
## [1] 1.570796
```

We can create variables using the assignment operator `<-`,

```
# create some variables  
x <- 5  
class <- 400  
hello <- "world"
```

and then use those variables in our functions.

```
# functions of variables  
log(x)
```

```
## [1] 1.609438
```

```
class^2
```

```
## [1] 160000
```

There are some rules for variable naming.

Variable names –

1. Can't start with a number.
2. Are case-sensitive.
3. Can be the name of a predefined internal function or letter in R (e.g., c, q, t, C, D, F, T, I). Try **not** to use these.
4. Cannot be reserved words that R (e.g., for, in, while, if, else, repeat, break, next).

## 1.1 Vectors

Variables can store more than one value, called a *vector*. We can create vectors using the combine (`c()`) function.

```
# store a vector
y <- c(1, 2, 6, 10, 17)
```

When we perform functions on our vector, the result is elementwise.

```
# elementwise function
y/2
```

```
## [1] 0.5 1.0 3.0 5.0 8.5
```

A vector must contain all values of the same type (i.e., numeric, integer, character, etc.).

We can also make sequences of numbers using either `:` or `seq()`.

```
# sequences
a <- 1:5
a
```

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

```
b <- seq(1, 5, by = 1)
b
```

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

We can **extract** values by index.

```
a[3]
```

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

Indexing is pretty powerful.

```
# indexing multiple items  
a[c(1, 3, 5)]
```

```
## [1] 1 3 5
```

```
a[1:3]
```

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

We can even tell R which elements we don't want.

```
a[-3]
```

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

And we can index by **logical** values. R has logicals built in using **TRUE** and **FALSE** (T and F also work, but can be overwritten). Logicals can result from a comparison using

- `<`: “less than”
- `>`: “greater than”
- `<=`: “less than or equal to”
- `>=`: “greater than or equal to”
- `==`: “is equal to”
- `!=`: “not equal to”

```
# indexing by vectors of logicals  
a[c(TRUE, TRUE, FALSE, FALSE, FALSE)]
```

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

```
# indexing by calculated logicals  
a < 3
```

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

```
a[a < 3]
```

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

We can combine elementwise logical vectors in the following way:

- `&` : elementwise AND
- `|` : elementwise OR

```
c(TRUE, TRUE, FALSE) | c(FALSE, TRUE, FALSE)
```

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

```
c(TRUE, TRUE, FALSE) & c(FALSE, TRUE, FALSE)
```

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

There are two more useful functions for looking at the start (head) and end (tail) of a vector.

```
head(a, 2)
```

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

```
tail(a, 2)
```

```
## [1] 4 5
```

We can also modify elements in a vector.

```
a[1] <- 0  
a[c(4, 5)] <- 100  
a
```

```
## [1] 0 2 3 100 100
```



As mentioned, elements of a vector must all be the same type. So, changing an element of a vector to a different type will result in all elements being converted to the *most general* type.

```
a
```

```
## [1] 0 2 3 100 100
```

```
a[1] <- ":-("
a
```

```
## [1] ":-(" "2" "3" "100" "100"
```

By changing a value to a string, all the other values were also changed.

There are many data types in R, numeric, integer, character (i.e., string), Date, and factor being the most common. We can convert between different types using the **as** series of functions.

```
as.character(b)
```

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

There are a whole variety of useful functions to operate on vectors. A couple of the more common ones are **length**, which returns the length (number of elements) of a vector, and **sum**, which adds up all the elements of a vector.

```
n <- length(b)
n
```

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

```
sum_b <- sum(b)
sum_b
```

```
## [1] 15
```

We can then create some statistics!

```
mean_b <- sum_b/n  
sd_b <- sqrt(sum((b - mean_b)^2)/(n - 1))
```

But, we don't have to.

```
mean(b)
```

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

```
sd(b)
```

```
## [1] 1.581139
```

```
summary(b)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
##         1         2         3         3         4         5
```

```
quantile(b, c(.25, .75))
```

```
## 25% 75%  
##   2   4
```

## 1.2 Data Frames

Data frames are the data structure you will (probably) use the most in R. You can think of a data frame as any sort of rectangular data. It is easy to conceptualize as a table, where each column is a vector. Recall, each vector must have the same data type *within* the vector (column), but columns in a data frame need not be of the same type. Let's look at an example!

```
# look at top 6 rows
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5         1.4         0.2   setosa
## 2           4.9         3.0         1.4         0.2   setosa
## 3           4.7         3.2         1.3         0.2   setosa
## 4           4.6         3.1         1.5         0.2   setosa
## 5           5.0         3.6         1.4         0.2   setosa
## 6           5.4         3.9         1.7         0.4   setosa
```

```
# structure of the object
str(iris)
```

```
## 'data.frame':   150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1
```

This is Anderson's Iris data set ([https://en.wikipedia.org/wiki/Iris\\_flower\\_data\\_set](https://en.wikipedia.org/wiki/Iris_flower_data_set)), available by default in R.

Some facts about data frames:

- Structured by rows and columns and can be indexed
- Each column is a variable of one type
- Column names or locations can be used to index a variable
- Advice for naming variables applies to naming columns
- Can be specified by grouping vectors of equal length as columns

Data frames are indexed (similarly to vectors) with `[ ]`.

- `df[i, j]` will select the element of the data frame in the `i`th row and the `j`th column.
- `df[i, ]` will select the entire `i`th row as a data frame
- `df[, j]` will select the entire `j`th column as a vector

We can use logicals or vectors to index as well.

```
iris[1, ]
```

```
## Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1          3.5          1.4          0.2  setosa
```

```
iris[, 1]
```

```
## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
## [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
## [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
## [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
## [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
## [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
iris[1, 1]
```

```
## [1] 5.1
```

We can also select columns by name in two ways.

```
iris$Species
```

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

```
## [25] setosa      setosa      setosa      setosa      setosa      setosa
## [31] setosa      setosa      setosa      setosa      setosa      setosa
## [37] setosa      setosa      setosa      setosa      setosa      setosa
## [43] setosa      setosa      setosa      setosa      setosa      setosa
## [49] setosa      setosa      versicolor  versicolor  versicolor  versicolor
## [55] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [61] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [67] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [73] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [79] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [85] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [91] versicolor  versicolor  versicolor  versicolor  versicolor  versicolor
## [97] versicolor  versicolor  versicolor  versicolor  virginica   virginica
## [103] virginica   virginica   virginica   virginica   virginica   virginica
## [109] virginica   virginica   virginica   virginica   virginica   virginica
## [115] virginica   virginica   virginica   virginica   virginica   virginica
## [121] virginica   virginica   virginica   virginica   virginica   virginica
## [127] virginica   virginica   virginica   virginica   virginica   virginica
## [133] virginica   virginica   virginica   virginica   virginica   virginica
## [139] virginica   virginica   virginica   virginica   virginica   virginica
## [145] virginica   virginica   virginica   virginica   virginica   virginica
## Levels: setosa versicolor virginica
```

```
iris[, "Species"]
```

```
## [1] setosa      setosa      setosa      setosa      setosa      setosa
## [7] setosa      setosa      setosa      setosa      setosa      setosa
## [13] setosa      setosa      setosa      setosa      setosa      setosa
## [19] setosa      setosa      setosa      setosa      setosa      setosa
## [25] setosa      setosa      setosa      setosa      setosa      setosa
## [31] setosa      setosa      setosa      setosa      setosa      setosa
## [37] setosa      setosa      setosa      setosa      setosa      setosa
## [43] setosa      setosa      setosa      setosa      setosa      setosa
## [49] setosa      setosa      versicolor  versicolor  versicolor  versico-
```

```

lor
## [55] versicolor versicolor versicolor versicolor versicolor versicolor
## [61] versicolor versicolor versicolor versicolor versicolor versicolor
## [67] versicolor versicolor versicolor versicolor versicolor versicolor
## [73] versicolor versicolor versicolor versicolor versicolor versicolor
## [79] versicolor versicolor versicolor versicolor versicolor versicolor
## [85] versicolor versicolor versicolor versicolor versicolor versicolor
## [91] versicolor versicolor versicolor versicolor versicolor versicolor
## [97] versicolor versicolor versicolor versicolor virginica virginica
## [103] virginica virginica virginica virginica virginica virginica
## [109] virginica virginica virginica virginica virginica virginica
## [115] virginica virginica virginica virginica virginica virginica
## [121] virginica virginica virginica virginica virginica virginica
## [127] virginica virginica virginica virginica virginica virginica
## [133] virginica virginica virginica virginica virginica virginica
## [139] virginica virginica virginica virginica virginica virginica
## [145] virginica virginica virginica virginica virginica virginica
## Levels: setosa versicolor virginica

```

To add columns, create a new vector that is the same length as other columns. We can append new column to the data frame using the `$` operator or the `[ ]` operators.

```

# make a copy of iris
my_iris <- iris

# add a column
my_iris$sepal_len_square <- my_iris$Sepal.Length^2
head(my_iris)

```

```

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species sepal_len_square
## 1           5.1         3.5         1.4         0.2   setosa           26.01
## 2           4.9         3.0         1.4         0.2   setosa           24.01
## 3           4.7         3.2         1.3         0.2   setosa           22.09
## 4           4.6         3.1         1.5         0.2   setosa           21.16
## 5           5.0         3.6         1.4         0.2   setosa           25.00
## 6           5.4         3.9         1.7         0.4   setosa           29.16

```

It's quite easy to subset a data frame.

```

my_iris[my_iris$sepal_len_square < 20, ]

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species sepal_l-

```

```
en_square
## 9      4.4      2.9      1.4      0.2 setosa      19.36
## 14     4.3      3.0      1.1      0.1 setosa      18.49
## 39     4.4      3.0      1.3      0.2 setosa      19.36
## 43     4.4      3.2      1.3      0.2 setosa      19.36
```

We'll see another way to do this in Lab 2.

We can create new data frames using the `data.frame()` function,

```
df <- data.frame(NUMS = 1:5,
                 lets = letters[1:5],
                 cols = c("green", "gold", "gold", "gold", "green"))
```

and we can change column names using the `names()` function.

```
names(df)
```

```
## [1] "NUMS" "lets" "cols"
```

```
names(df)[1] <- "nums"
```

```
df
```

```
##  nums lets  cols
##  1     1    a green
##  2     2    b  gold
##  3     3    c  gold
##  4     4    d  gold
##  5     5    e green
```

There are other data structures available to you in R, namely lists and matrices. We will not cover these in the notes, but I encourage you to read more about them (<https://faculty.nps.edu/sebuttre/home/R/lists.html> and <https://faculty.nps.edu/sebuttre/home/R/matrices.html>).

## 1.3 Basic Programming

We will cover three basic programming ideas: functions, conditionals, and loops.

### 1.3.1 Functions

We have used many functions that are already built into R already. For example – `exp()`, `log()`, `sin()`, `rep()`, `seq()`, `head()`, `tail()`, etc.

But what if we want to use a function that doesn't exist?

**We can write it!**

Idea: We want to avoid repetitive coding because errors will creep in. Solution: Extract common core of the code, wrap it in a function, and make it reusable.

The basic structure for writing a function is as follows:

- Name
- Input arguments (including names and default values)
- Body (code)
- Output values

```
# we store a function in a named value  
# function is itself a function to create functions!  
# we specify the inputs that we can use inside the function  
# we can specify default values, but it is not necessary  
name <- function(input = FALSE) {  
  # body code goes here  
  
  # return output vaues  
  return(input)  
}
```

Here is a more realistic first example:

```
my_mean <- function(x) {  
  sum(x)/length(x)  
}
```

Let's test it out.



```
my_mean(1:15)
```

```
## [1] 8
```

```
my_mean(c(1:15, NA))
```

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

Some advice for function writing:

1. Start simple, then extend.
2. Test out each step of the way.
3. Don't try too much at once.

### 1.3.2 Conditionals

Conditionals are functions that control the flow of analysis. Conditionals determine if a specified condition is met (or not), then direct subsequent analysis or action depending on whether the condition is met (or not).

```
if(condition) {
  # Some code that runs if condition is TRUE
} else {
  # Some code that runs if condition is TRUE
}
```

- `condition` is a length one logical value, i.e. either `TRUE` or `FALSE`
- We can use `&` and `|` to combine several conditions
- `!` negates condition

For example, if we wanted to do something with `na.rm` from our function,

```
if(na.rm) x <- na.omit(x) # na.omit is a function that removes NA
values
```

might be a good option.

### 1.3.3 Loops

Loops (and their cousins the `apply()` function) are useful when we want to repeat the same block of code many times. Reducing the amount of typing we do can be nice, and if we have a lot of code that is essentially the same we can take advantage of looping. R offers several loops: `for`, `while`, `repeat`.

`for` loops will run through a specified index and perform a set of code for each value of the indexing variable.

```
for(i in index values) {
  # block of code
  # can print values also
  # code in here will most likely depend on i
}
```

```
for(i in 1:3) {
  print(i)
}
```

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

```
for(species in unique(iris$Species)) {
  subset_iris <- iris[iris$Species == species,]
  avg <- mean(subset_iris$Sepal.Length)
  print(paste(species, avg))
}
```

```
## [1] "setosa 5.006"
## [1] "versicolor 5.936"
## [1] "virginica 6.588"
```

`while` loops will run until a specified condition is no longer true.

```
condition <- TRUE
while(condition) {
  # do stuff
}
```

```
# don't forget to eventually set the condition to false  
# in the toy example below I check if the current seconds is  
divisible by 5  
time <- Sys.time()  
if(as.numeric(format(time, format = "%S")) %% 5 == 0) condition <-  
FALSE  
}  
print(time)
```

```
## [1] "2020-08-24 15:14:55 MDT"
```

```
# we can also use while loops to iterate  
i <- 1  
while (i <= 5) {  
  print(i)  
  i <- i + 1  
}
```

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

## 1.4 Packages

Commonly used R functions are installed with base R.

R packages containing more specialized R functions can be installed freely from CRAN servers using function `install.packages()`.

After packages are installed, their functions can be loaded into the current R session using the function `library()`.

Packages are contributed by R users just like you!

We will use some great packages in this class. Feel free to venture out and find your favorites (google R package + what you're trying to do to find more packages).

## 1.5 Additional resources

You can get help with R functions within R by using the `help()` function, or typing `?` before a function name.

Stackoverflow can be helpful – if you have a question, maybe somebody else has already asked it (<https://stackoverflow.com/questions/tagged/r>).

R Reference Card (<https://cran.r-project.org/doc/contrib/Short-refcard.pdf>)

Useful Cheatsheets (<https://www.rstudio.com/resources/cheatsheets/>)

R for Data Science (<https://r4ds.had.co.nz>)

Advanced R (<https://adv-r.hadley.nz>)

## 2 Rmarkdown

**Markdown** is a particular type of markup language that is designed to produce documents from text.

Markdown is becoming a standard. Many websites will generate HTML from Markdown (e.g. GitHub, Stack Overflow, reddit, ...) and this course website is written in markdown as well.

Markdown is easy for humans to read and write.

```
*italic*  
**bold**
```

```
# Header 1  
## Header 2  
### Header 3
```

```
* Item 1  
* Item 2  
  + Item 2a  
  + Item 2b
```

```
1. Item 1  
2. Item 2  
3. Item 3  
  + Item 3a  
  + Item 3b
```

```
[linked phrase](http://example.com)
```

```
A friend once said:
```

```
> It's always better to give  
> than to receive.
```

**Rmarkdown** is an authoring format that lets you incorporate the results from R code in your documents.

You no longer have to copy/paste plots into your homework!

Documents built from Rmarkdown are fully reproducible, i.e. they are automatically re-generated whenever embedded R code changes.

To include an R chunk in an Rmarkdown document, you use backticks.

In order to create a new Rmarkdown document in RStudio, File > New File > R markdown.

There are many options that can affect the aesthetics of the resulting document and the results and appearance of R chunks. For a list of chunk options, see <https://yihui.name/knitr/options/>. Here are some useful ones:

- `echo` - should the code be printed?
- `message` and `warning` - should message and warnings be printed?
- `eval` - should the code be evaluated?
- `fig.height` & `fig.width` - figure height and width
- `fig.cap` - figure caption

## 2.1 Additional resources

Documentation and cheat sheets (<https://rmarkdown.rstudio.com>)

R Markdown: The Definitive Guide (<https://bookdown.org/yihui/rmarkdown/>)