

# Introduction to R and Data Concepts

## Introduction to R and Data Concepts

### Goals

- Gain some familiarity and comfort with rstudio
- Review how to assign variables
- Learn about and use functions
- Learn about and use vectors
- Write code to work with a data set, and calculate some descriptive statistics

### Arithmetic

The grey rectangle below is a "code chunk". Everything within the grey area is interpreted as R code. To run the code, click the green triangle in the upper-right corner.

In this example, R can perform basic math:

```
10+7
```

```
[1] 17
```

Now it's your turn. Enter code below to subtract ten from twenty-two:

```
22-10
```

```
[1] 12
```

### Assigning Variables

We'll be working a lot with variables throughout this semester. A variable is a name you give to some value. The value could be a single number, a word, a bunch of words, an entire data set, etc.

Most scripting languages use the "=" sign to assign a value to a variable, but R uses "<-".

```
# assigns 10 to x
```

```
x<-10
```

It's important to note that creating a new variable using code above doesn't give you any output. Often it's a good idea to print your variable to the screen, just to confirm it worked the way you intended:

```
# prints x
```

```
x
```

```
[1] 10
```

```
#Anything preceded by a "#" is a "comment". It does not get executed as code.  
#Comments can be super helpful to provide info on your code.
```

Now it's your turn. Create a variable "y", set it equal to 7+9, and then print it out:

```
y<-7+9
```

## Functions

Coding languages, including R, have functions that help you quickly execute common tasks. Functions typically take the form of:

```
functionName(argument1, argument2, etc....)
```

*Arguments* are the inputs you send to a function, so it has all the information it needs to perform its operation.

For example, the function `sqrt(number)` takes the square root of a number. This lets us quickly compute the answer, rather than having to write the formula for a square root.

```
sqrt(9)
```

```
[1] 3
```

YOUR TURN: In the chunk below, create a variable `z`, set it equal to the square root of 90, and print it out:

```
z<-sqrt(90)
```

```
z
```

```
[1] 9.486833
```

One nice thing about Rstudio is that you can readily access documentation for functions by using the "help" command:

```
help(sqrt)
```

The documentation appears in the lower right window in the "help" tab.

One key question: how do you know what functions exist, and what they do?

Answer: you Google what you're trying to do! In the case of R, you might search "How do I do 'x' in R?"

Let's say you are interested in calculating the absolute value (positive distance from zero) of -35 in R. Take a moment with your group/neighbors, and try to find the answer by searching the internet. In the code chunk below, use the function you found to compute this calculation:

```
# compute the absolute value of -35  
  
abs(-35)
```

```
[1] 35
```

## Vectors

So far we've created variables that have single values (e.g. `x<-7`), but there are often cases where we need to assign *multiple values* to a variable. These types of variables are called *vectors*.

In order to create a vector, you can use the "c" function (c stands for "combine"). Here's an example:

```
myFirstVector<-c(3,7,1,10)  
  
myFirstVector
```

```
[1] 3 7 1 10
```

Now it's your turn. Create a vector called `mySecondVector`, assign the values 8, -11, 100, 35 to it, and print it to the screen:

```
# use the "c" function to create mySecondVector:  
  
mySecondVector<-c(8,-11,100, 35)  
  
mySecondVector
```

```
[1] 8 -11 100 35
```

Before moving on, let's talk a little about variable naming conventions. We started out using `x`, `y`, and `z` when learning about variables. That technically works, but it's better practice to be more descriptive in your variable names. The examples above and below use a syntax called "camel case". This allows you to string words together without spaces, but preserves quick readability. From this point on, we're going to create variables with camel case - you should too!





variable using the `read_csv` function:

```
#load the data
teamAntarcticaData<-read_csv("teamAntarcticaData.csv")
```

Rows: 75 Columns: 12

— Column specification —

Delimiter: ",",

chr (7): Timestamp, school, swim, animals, parkaColor, teamFlag, distance

dbl (5): fishing, cold, remote, bedsideManner, cooking

- i Use ``spec()`` to retrieve the full column specification for this data.
- i Specify the column types or set ``show_col_types = FALSE`` to quiet this message.

```
#print to screen
teamAntarcticaData
```

# A tibble: 75 × 12

	Timestamp	school	fishing	swim	cold	animals	remote	parkaColor	teamFlag
	<chr>	<chr>	<dbl>	<chr>	<dbl>	<chr>	<dbl>	<chr>	<chr>
1	8/30/2022 16:0...	Unive...	1	Yes	4	Yes	4	Gold	Penguin
2	8/30/2022 16:0...	Unive...	2	Yes	4	Yes	5	Blue	Bear
3	8/30/2022 16:0...	Unive...	2	Yes	4	Yes	3	Green	Penguin
4	8/30/2022 16:0...	Unive...	1	Yes	1	Yes	1	Blue	Seal
5	8/30/2022 16:0...	Unive...	1	Yes	3	Yes	3	White	Sea Spi...
6	8/30/2022 16:0...	Unive...	1	Yes	3	Yes	3	hot pink	Penguin
7	8/30/2022 16:0...	Unive...	1	Yes	2	Yes	3	Blue	Sea Spi...
8	8/30/2022 16:0...	Unive...	2	Yes	2	Yes	4	Blue	Penguin
9	8/30/2022 16:0...	Unive...	2	Yes	2	Yes	5	White	Bear
10	8/30/2022 16:0...	Unive...	5	Yes	5	Yes	5	Blue	Penguin

# i 65 more rows

# i 3 more variables: distance <chr>, bedsideManner <dbl>, cooking <dbl>

```
#It's also very useful to click the variable name of a dataset in the Environment
#This will open the data in a new tab, and is very easy to read.
```

Earlier in this exercise we looked at the array of responses for both fishing and cooking aptitude, though in both cases the vectors were hand-typed (by me). A much more common way to acquire, and then use, a vector of data is to directly query the data set. You can get a vector (a.k.a. column) of data by using the following syntax:

`dataSet$columnName`

Let's get all responses for fishing aptitude directly from the data set:

```
fishing<-teamAntarcticaData$fishing
```

And just like before, we can calculate the mean, median, and standard deviation:

```
mean(fishing)
```

```
[1] 2.146667
```

```
median(fishing)
```

```
[1] 2
```

```
sd(fishing)
```

```
[1] 1.248711
```

Now it's your turn:

Use the data set to get the column values for tolerance of cold (hint: after typing the \$, use auto-complete to select the column name). Calculate its mean, median, and standard deviation.

```
# create a vector that contains the column values for cold tolerance  
cold<-teamAntarcticaData$cold  
  
#calculate the mean  
mean(cold)
```

```
[1] 3.373333
```

```
#calculate the median  
median(cold)
```

```
[1] 3
```

```
# calculate the standard deviation  
sd(cold)
```

```
[1] 0.9969322
```

Now do the same for comfort level with being in a remote location:

```
# create a vector that contains the column values for comfort level with remote l  
remote<-teamAntarcticaData$remote
```

```
#calculate the mean
```

```
mean(remote)
```

```
[1] 3.28
```

```
#calculate the median
```

```
median(remote)
```

```
[1] 3
```

```
# calculate the standard deviation
```

```
sd(remote)
```

```
[1] 1.133757
```

Now create a vector to get the responses for parka color. How is this data different from the other examples we've seen? What can we learn from the data?

```
parkas<-teamAntarcticaData$parkaColor
```

```
parkas
```

```
[1] "Gold"           "Blue"           "Green"
[4] "Blue"           "White"          "hot pink"
[7] "Blue"           "Blue"          "White"
[10] "Blue"           "purple"        "White"
[13] "Green"          "Black"         "White"
[16] "Orange"         "Orange"        "Orange"
[19] "White"          "Blue"          "Black"
[22] "Blue"           "Pink, if possible" "green"
[25] "Black"          "Black"         "White"
[28] "Black"          "White"         "Blue"
[31] "Green"          "Blue"          "Black"
[34] "Blue"           "White"         "Blue"
[37] "Green"          "Blue"          "Black"
[40] "Blue"           "Blue"          "Black"
[43] "Blue"           "Black"         "Blue"
[46] "Orange"         "Orange"        "Blue"
[49] "Orange"         "Black"         "Black"
[52] "Pink"           "Baby Pink"     "Blue"
[55] "Lavender/purple" "White"         "Black"
[58] "Black"          "Blue"          "Black"
[61] "Orange"         "Blue"          "Blue"
[64] "Blue"           "Orange"        "White"
```

[67]	"Orange"	"Black"	NA
[70]	"Black"	"White"	"Black"
[73]	"Black"	"Orange"	"Purple"

Next up: [practice-problems-1.2.qmd](#)