

Computational Thinking1

Ward, A., Miller, P.

Activity 7: Computational thinking 1: functions and iteration

Download Packages

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.6
v forcats    1.0.0      v stringr    1.5.2
v ggplot2    4.0.1      v tibble     3.3.0
v lubridate  1.9.4      v tidyr      1.3.2
v purrr      1.1.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()     masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
```

```
library(here)
```

here() starts at /Users/Abbie1/Documents/Repositories/BI0E 176 DataScience4EEB/BI0E276_CompT

1. Functions

Working with single values as input

```
# Create a new function called add_one
# x will be the only input to the function
add_one <- function(x){

  # Add x and 1 together, store as the object "output"
  output <- x + 1

  # Print out whatever is stored in "output"
  return(output)

}
```

Using our function

```
# Supply 10 to our function
add_one(x = 10)
```

```
[1] 11
```

More complicated function

```
# Create a new function called add_together
# x and y will be the two arguments to the function
add_together <- function(x, y){

  # Add x and y together, store as the object "output"
  output <- x + y

  # Print out whatever is stored in "output"
  return(output)

}
```

Q1.1 Feed our new function two numbers

```
add_together(x=3, y=5)
```

```
[1] 8
```

Q1.2 Feed our new function a number and a character string

```
add_together(3, "five")
```

“Error in x + y : non-numeric argument to binary operator”

Access source code of function

```
add_together
```

```
function (x, y)
{
  output <- x + y
  return(output)
}
```

Q1.3 Create your own function!

```
#Create Math_time
math_time <- function(x,y,z){
  output <- (x-y)^2 / z
  return(output)
}

#Run Math_time
math_time(5,2,9)
```

```
[1] 1
```

Output = 1

Working with vectors as input

```
# Create a function called lbs_to_kg that takes a data object 'weights' as input
lbs_to_kg <- function(weights){

  # Multiply weights by 0.454, store as the object "output"
  output <- weights*0.454

  # Print out whatever is stored in "output"
  return(output)

}
```

```
#Create the vector
bison <- c(1000, 800, 1200, 1400)
```

```
#Feed it into the function
lbs_to_kg(weights = bison)
```

```
[1] 454.0 363.2 544.8 635.6
```

Q1.4 Calculate deviation from a mean

```
#Creating deviation function
deviation <- function(my_vector) {
  mean <- mean(my_vector)
  output <- mean - my_vector
  return(output)
}
```

```
#Bison SD
deviation(my_vector = bison)
```

```
[1] 100 300 -100 -300
```

2. Iteration

```
# Look at the first 6 rows of iris
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

Q2.1: What are the units that the sepal and petal columns are measured in?

```
?iris
#All measurements in centimeters
```

```
#Can do this for each column
iris %>%
  group_by(Species) %>%
  summarize(Sepal.Length = mean(Sepal.Length),
            Sepal.Width = mean(Sepal.Width),
            Petal.Length = mean(Petal.Length),
            Petal.Width = mean(Petal.Width))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>          <dbl>         <dbl>         <dbl>         <dbl>
1 setosa          5.01          3.43          1.46          0.246
2 versicolor      5.94          2.77          4.26          1.33
3 virginica       6.59          2.97          5.55          2.03
```

Instead we can use across!

```
#Using Across
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = c(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width),
                      .fns = mean))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>          <dbl>        <dbl>        <dbl>        <dbl>
1 setosa         5.01          3.43          1.46          0.246
2 versicolor     5.94          2.77          4.26          1.33
3 virginica      6.59          2.97          5.55          2.03
```

```
#Using coloumn from length to pedal width
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = Sepal.Length:Petal.Width,
                    .fns = mean))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>          <dbl>        <dbl>        <dbl>        <dbl>
1 setosa         5.01          3.43          1.46          0.246
2 versicolor     5.94          2.77          4.26          1.33
3 virginica      6.59          2.97          5.55          2.03
```

```
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = 1:4,
                    .fns = mean))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>          <dbl>        <dbl>        <dbl>        <dbl>
1 setosa         5.01          3.43          1.46          0.246
2 versicolor     5.94          2.77          4.26          1.33
3 virginica      6.59          2.97          5.55          2.03
```

Using everything()

```
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = everything(),
                    .fns = mean))
```

```
# A tibble: 3 x 5
```

	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	setosa	5.01	3.43	1.46	0.246
2	versicolor	5.94	2.77	4.26	1.33
3	virginica	6.59	2.97	5.55	2.03

Q2.2 Summarize to calculate the median across all columns

```
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = Sepal.Length:Petal.Width,
                    .fns = median))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>         <dbl>         <dbl>         <dbl>         <dbl>
1 setosa         5           3.4           1.5           0.2
2 versicolor    5.9           2.8           4.35          1.3
3 virginica     6.5           3            5.55          2
```

#Changed .fns to median

```
#where() to select columns
iris %>%
  group_by(Species) %>%
  summarize(across(.cols = where(is.numeric),
                    .fns = mean))
```

```
# A tibble: 3 x 5
  Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
  <fct>         <dbl>         <dbl>         <dbl>         <dbl>
1 setosa         5.01           3.43           1.46           0.246
2 versicolor    5.94           2.77           4.26           1.33
3 virginica     6.59           2.97           5.55           2.03
```

Q2.3 Summarize to calculate the mean across all numeric columns in the cereal data

```
#read in cereal df
cereal <- read_csv("./data/cereal.csv")
```

Rows: 77 Columns: 16

-- Column specification -----

Delimiter: ","

chr (3): name, mfr, type

dbl (13): calories, protein, fat, sodium, fiber, carbo, sugars, potass, vita...

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.

```
#Mean across all numeric columns
cereal %>%
  group_by(mfr) %>%
  summarize(across(.cols = where(is.numeric),
                      .fns = mean))
```

A tibble: 7 x 14

	mfr	calories	protein	fat	sodium	fiber	carbo	sugars	potass	vitamins	shelf
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Americ~	100	4	1	0	0	16	3	95	25	2
2	Genera~	111.	2.32	1.36	200.	1.27	14.7	7.95	85.2	35.2	2.14
3	Kellog~	109.	2.65	0.609	175.	2.74	15.1	7.57	103.	34.8	2.35
4	Nabisco	86.7	2.83	0.167	37.5	4	16	1.83	121.	8.33	1.67
5	Post	109.	2.44	0.889	146.	2.78	13.2	8.78	114.	25	2.44
6	Quaker~	95	2.62	1.75	92.5	1.34	10	5.25	74.4	12.5	2.38
7	Ralsto~	115	2.5	1.25	198.	1.88	17.6	6.12	89.2	25	2

i 3 more variables: weight <dbl>, cups <dbl>, rating <dbl>

For-loops

Loop through a simple equation

```
#most basic loop
for (i in 1:5) {
  # Print out whatever the value of i is
  print(i)
}
```



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

Multiply i value by 2

```
for (i in 1:5) {
  print(i*2)
}
```

```
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
```

Q2.4 Create a for loop that iterates from 1 to 10 and squares the value of the index i

```
for (i in 1:10) {
  print (i^2)
}
```

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
[1] 49
[1] 64
[1] 81
[1] 100
```

Simulate data using a for-loop

```

N0 = 100 #initial population size

years = 20 #number of years into the future

N = vector(length = years) # create an empty vector to store pop. sizes

N[1] = N0 #initial population size should be the first N

lambda = 1.2 #growth rate

```

```

#Printing the full vector
print(N)

```

```

[1] 100  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[20]  0

```

```

#Look at one element in vector
N[5]

```

```

[1] 0

```

```

# For every year t in 2 through 20 (remember, "years" also equals 20), apply the following equation
for (t in 2:20) {
  N[t] = N[t - 1] * lambda # Apply the equation
}

```

Check N

```

N

```

```

[1] 100.0000 120.0000 144.0000 172.8000 207.3600 248.8320 298.5984
[8] 358.3181 429.9817 515.9780 619.1736 743.0084 891.6100 1069.9321
[15] 1283.9185 1540.7022 1848.8426 2218.6111 2662.3333 3194.8000

```

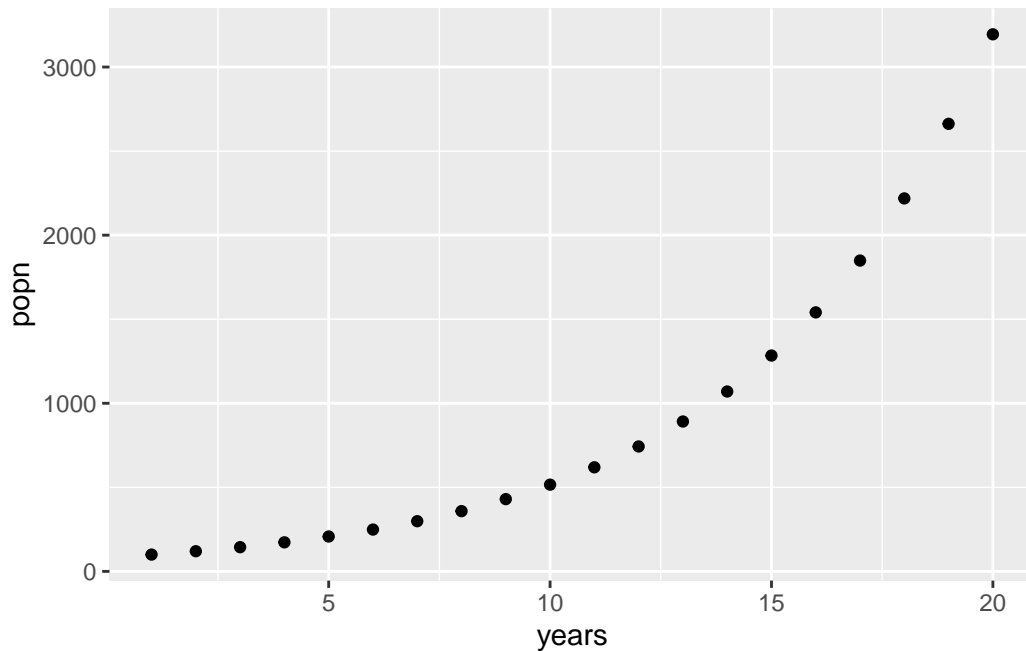
```

# Store the data output as a dataframe for plotting
popn_data <- tibble(years = 1:years, # Make the years column = 1, 2, 3, ..., 20
                    popn = N) # Make the population column the corresponding population vector

```

Plot the data

```
# Now plot the data with years on the x axis and population on the y
popn_data %>%
  ggplot(aes(x = years, y = popn)) +
  geom_point()
```



Q2.5a Rerun the for loop with the following parameters:

```
#Changing intial parameters
N0 = 300 #initial population size

years = 50 #number of years into the future

N = vector(length = years) # create an empty vector to store pop. sizes

N[1] = N0 #initial population size should be the first N

lambda = 0.95 #growth rate
```

```
#Rerun loop for 50
for (t in 2:50) {
```

```

  N[t] = N[t - 1] * lambda # Apply the equation
}

```

```

#Look at N now
N

```

```

[1] 300.00000 285.00000 270.75000 257.21250 244.35187 232.13428 220.52757
[8] 209.50119 199.02613 189.07482 179.62108 170.64003 162.10803 154.00262
[15] 146.30249 138.98737 132.03800 125.43610 119.16430 113.20608 107.54578
[22] 102.16849 97.06006 92.20706 87.59671 83.21687 79.05603 75.10323
[29] 71.34807 67.78066 64.39163 61.17205 58.11345 55.20777 52.44738
[36] 49.82502 47.33376 44.96708 42.71872 40.58279 38.55365 36.62596
[43] 34.79467 33.05493 31.40219 29.83208 28.34047 26.92345 25.57728
[50] 24.29841

```

```

# Store the data output as a dataframe for plotting
popn_data2 <- tibble(years = 1:years, # Make the years column = 1, 2, 3, ..., 50
                     popn = N) # Make the population column the corresponding population vect

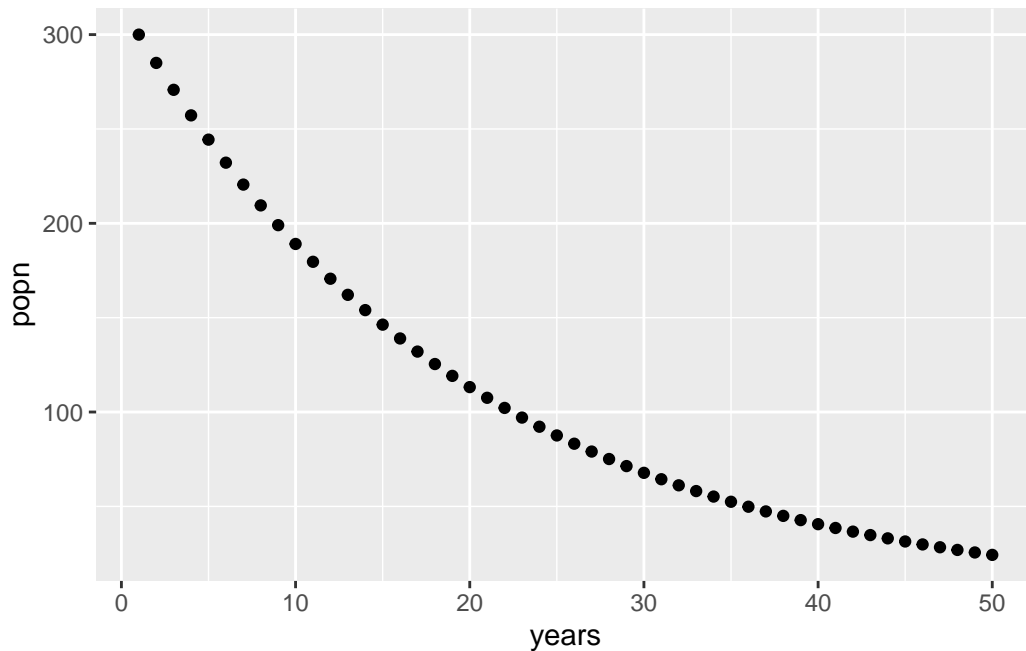
```

Q2.5b Plot the data and interpret

```

popn_data2 %>%
  ggplot(aes(x = years, y = popn)) +
  geom_point()

```



Changing the initial population size and the population growth rate causes a exponential decline to the overall population size.

Loop through data frames

```
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

```
#Extract row 1
iris[1,]
```

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

```
#Extract column 3
iris[,3]
```

```
[1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
[19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
[37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
[55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
[73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
[91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
[109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
[127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
[145] 5.7 5.2 5.0 5.2 5.4 5.1
```

```
#Extract exact cell
iris[1,3]
```

```
[1] 1.4
```

Printing out all of the values for each column 1-5

```
for (i in 1:5) {

  # This prints out a statement saying "Here's column i",
  #but the i gets replaced with the number that it's currently at
  print(paste("Here's column",i))

  # This prints out column i
  print(iris[,i])
}
```

```
[1] "Here's column 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
```

```

[1] "Here's column 2"
[1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9 3.5
[19] 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2 3.1 3.2
[37] 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2 3.2 3.1 2.3
[55] 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7 2.2 2.5 3.2 2.8
[73] 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0 3.4 3.1 2.3 3.0 2.5
[91] 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7 3.0 2.9 3.0 3.0 2.5 2.9
[109] 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6 2.2 3.2 2.8 2.8 2.7 3.3 3.2
[127] 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2
[145] 3.3 3.0 2.5 3.0 3.4 3.0

[1] "Here's column 3"
[1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2 1.5 1.3 1.4
[19] 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6 1.6 1.5 1.5 1.4 1.5 1.2
[37] 1.3 1.4 1.3 1.5 1.3 1.3 1.3 1.6 1.9 1.4 1.6 1.4 1.5 1.4 4.7 4.5 4.9 4.0
[55] 4.6 4.5 4.7 3.3 4.6 3.9 3.5 4.2 4.0 4.7 3.6 4.4 4.5 4.1 4.5 3.9 4.8 4.0
[73] 4.9 4.7 4.3 4.4 4.8 5.0 4.5 3.5 3.8 3.7 3.9 5.1 4.5 4.5 4.7 4.4 4.1 4.0
[91] 4.4 4.6 4.0 3.3 4.2 4.2 4.2 4.3 3.0 4.1 6.0 5.1 5.9 5.6 5.8 6.6 4.5 6.3
[109] 5.8 6.1 5.1 5.3 5.5 5.0 5.1 5.3 5.5 6.7 6.9 5.0 5.7 4.9 6.7 4.9 5.7 6.0
[127] 4.8 4.9 5.6 5.8 6.1 6.4 5.6 5.1 5.6 6.1 5.6 5.5 4.8 5.4 5.6 5.1 5.1 5.9
[145] 5.7 5.2 5.0 5.2 5.4 5.1

[1] "Here's column 4"
[1] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 0.2 0.2 0.1 0.1 0.2 0.4 0.4 0.3
[19] 0.3 0.3 0.2 0.4 0.2 0.5 0.2 0.2 0.4 0.2 0.2 0.2 0.2 0.4 0.1 0.2 0.2 0.2
[37] 0.2 0.1 0.2 0.2 0.3 0.3 0.2 0.6 0.4 0.3 0.2 0.2 0.2 0.2 1.4 1.5 1.5 1.3
[55] 1.5 1.3 1.6 1.0 1.3 1.4 1.0 1.5 1.0 1.4 1.3 1.4 1.5 1.0 1.5 1.1 1.8 1.3
[73] 1.5 1.2 1.3 1.4 1.4 1.7 1.5 1.0 1.1 1.0 1.2 1.6 1.5 1.6 1.5 1.3 1.3 1.3
[91] 1.2 1.4 1.2 1.0 1.3 1.2 1.3 1.3 1.1 1.3 2.5 1.9 2.1 1.8 2.2 2.1 1.7 1.8
[109] 1.8 2.5 2.0 1.9 2.1 2.0 2.4 2.3 1.8 2.2 2.3 1.5 2.3 2.0 2.0 1.8 2.1 1.8
[127] 1.8 1.8 2.1 1.6 1.9 2.0 2.2 1.5 1.4 2.3 2.4 1.8 1.8 2.1 2.4 2.3 1.9 2.3
[145] 2.5 2.3 1.9 2.0 2.3 1.8

[1] "Here's column 5"
[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 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

```

Take mean of each column

```

for (i in 1:4) {

  # This prints out a statement saying "Here's column i", but the i gets replaced with the number
  print(paste("Here's column",i))

  # This prints out column i
  print(mean(iris[,i]))
}

```

```

[1] "Here's column 1"
[1] 5.843333
[1] "Here's column 2"
[1] 3.057333
[1] "Here's column 3"
[1] 3.758
[1] "Here's column 4"
[1] 1.199333

```

Now with across()

```

iris %>%
  summarize(across(.cols = 1:4,
                    .fns = mean))

```

```

Sepal.Length Sepal.Width Petal.Length Petal.Width
1      5.843333      3.057333      3.758      1.199333

```


Q2.6 Which do you prefer?

For this example specifically, I think the across function is much more straight forward. The for loop is helpful in many other instance, but for something so simple, the across functions seems to make more sense.

Creating a vector of column names

```
for (i in 1:4) {  
  # Fetch the column names of the dataframe, store in a vector "names"  
  names <- colnames(iris)  
  
  # Print out the "i"th element of the vector to print alongside the output  
  print(names[i])  
  
  # This prints out column i  
  print(mean(iris[,i]))  
}
```

```
[1] "Sepal.Length"  
[1] 5.843333  
[1] "Sepal.Width"  
[1] 3.057333  
[1] "Petal.Length"  
[1] 3.758  
[1] "Petal.Width"  
[1] 1.199333
```

Q2.7 Annotate this code with what you think it's doing

```
# Store a vector of unique species names from the Species column of Iris  
spp_names <- unique(iris$Species)  
  
# Creating a for loop through all values 1 through the length of the spp_names vector  
for (i in 1:length(spp_names)) {  
  
  filt_data <- iris %>%  
    # filters the iris df where Species is equal to one of each of the species names (since :  
    filter(Species == spp_names[i])  
  
  # naming a new object called "plot" to begin creating a plot
```

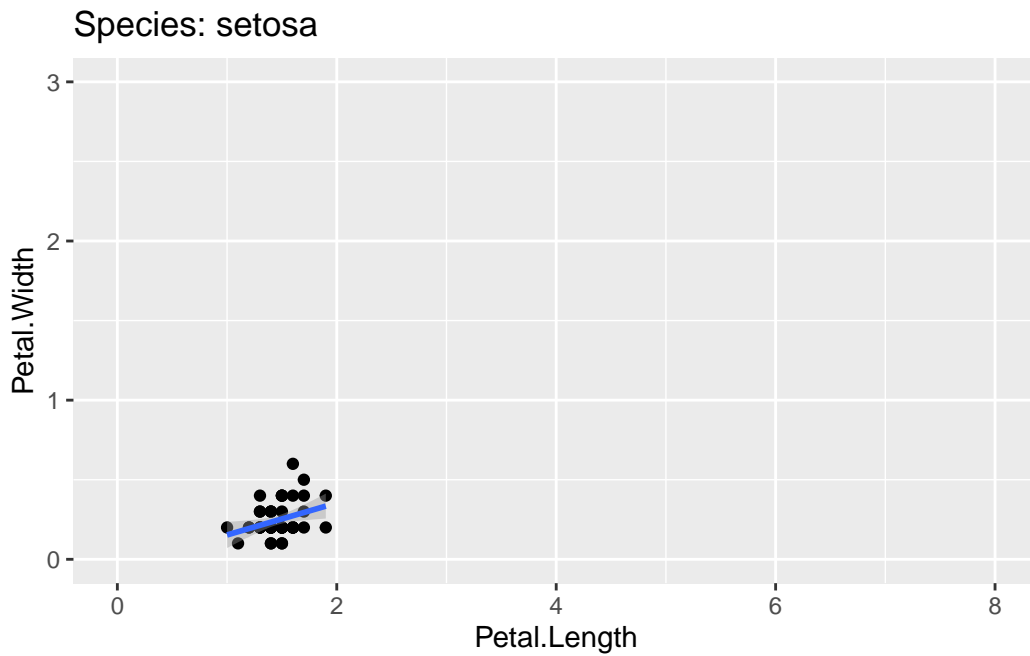
```

plot <- filt_data %>%
  # using ggplot to begin mapping our aesthetics of petal length and petal width
  ggplot(aes(x = Petal.Length,
             y = Petal.Width)) +
  # telling ggplot to plot point
  geom_point() +
  # telling ggplot to plot a regression line
  geom_smooth(method = "lm") +
  # giving limits to the x and y axes
  lims(x = c(0,8),
       y = c(0,3)) +
  # Title of the plot will be Species: and then each of the species from the spp_names df
  ggtitle(paste("Species:", spp_names[i]))

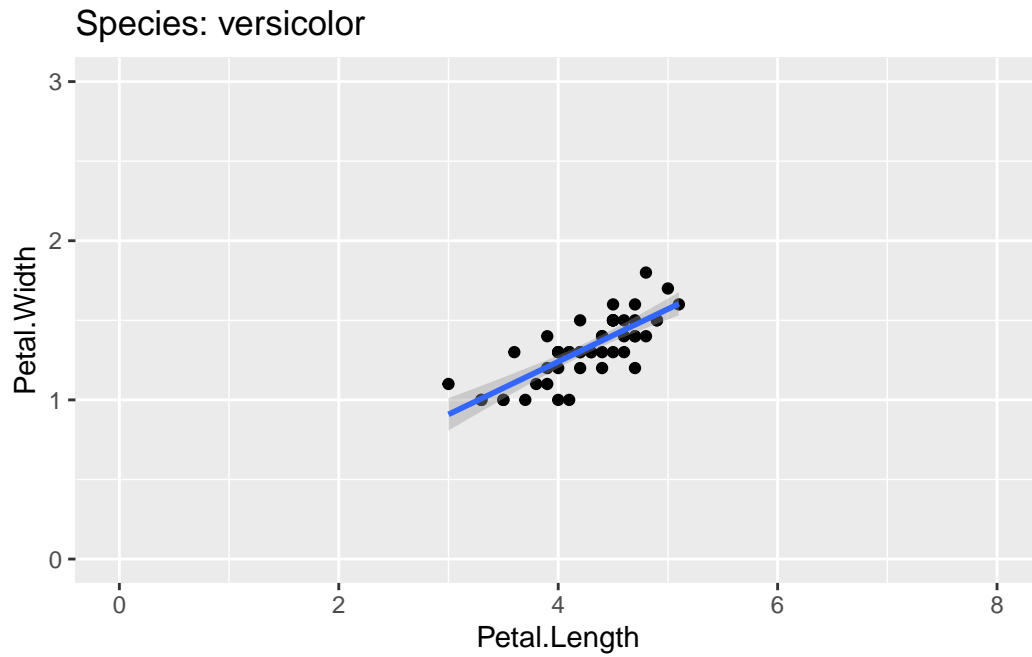
# printing the plot
print(plot)
}

```

`geom_smooth()` using formula = 'y ~ x'



`geom_smooth()` using formula = 'y ~ x'



``geom_smooth()`` using formula = 'y ~ x'

