

Computational Thinking 1

Paige G and Leah W

Libraries

```
library(tidyverse)
```

Warning: package 'ggplot2' was built under R version 4.4.3

Warning: package 'purrr' was built under R version 4.4.3

```
-- Attaching core tidyverse packages ----- tidyverse 2.0.0 --
v dplyr      1.1.4      v readr      2.1.5
v forcats    1.0.0      v stringr    1.5.1
v ggplot2    4.0.1      v tibble     3.2.1
v lubridate  1.9.4      v tidyr      1.3.1
v purrr      1.2.1
-- 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)
```

Warning: package 'here' was built under R version 4.4.3

here() starts at C:/Users/jgard/OneDrive/Paige's stuff 12.20/Graduate School/UCSC/Project pl

Question 1.1

```
# 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)

}

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

```
[1] 8
```

Question 1.2

```
add_together(x=3, y="five")
```

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

The error code is telling us that it's a numeric structure value, but it got a string instead.

Question 1.3

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

math_time(x=5,y=2,z=9)
```

```
[1] 1
```

Question 1.4

```
bison <- c(1000, 800, 1200, 1400)

deviation <- function(weights){
  mean<- mean(weights)
  sd <- weights-mean
  return(sd)
}

deviation(bison)
```

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

Iteration

Across function

```
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 |

Q 2.1

```
help(iris)
```

```
starting httpd help server ... done
```

The measurements are in centimeters.

Q2.2

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

Question 2.3

```
cereal <- read.csv(here("data/cereal.csv"))
colnames(cereal)
```

```
[1] "name"      "mfr"      "type"      "calories" "protein"  "fat"
[7] "sodium"    "fiber"    "carbo"     "sugars"   "potass"   "vitamins"
[13] "shelf"     "weight"   "cups"      "rating"
```

```
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>

Q2.4

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

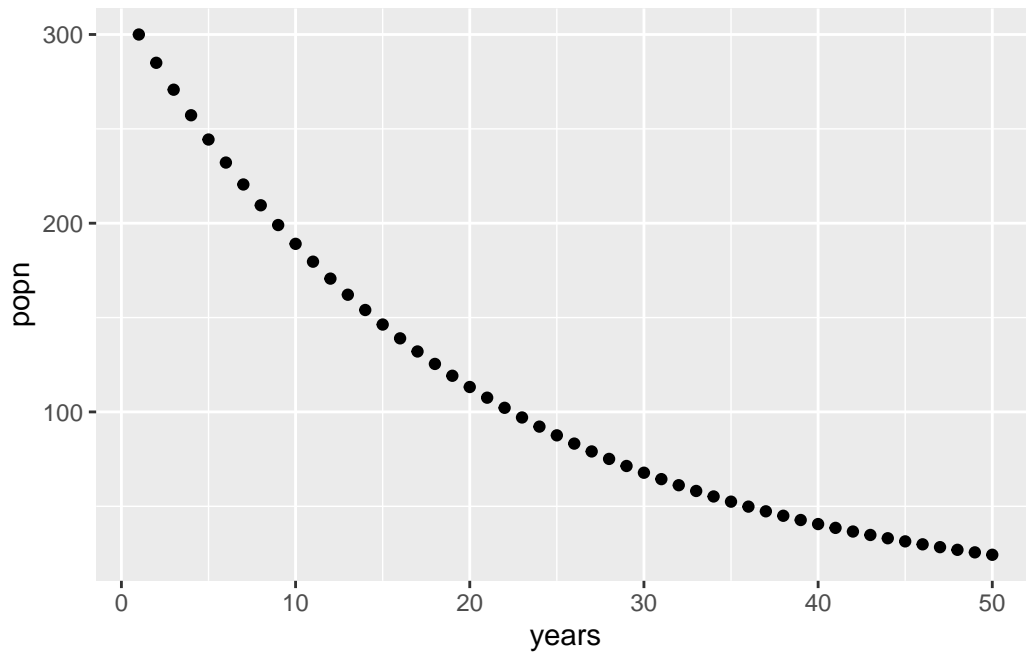
```
[1] 1  
[1] 1.414214  
[1] 1.732051  
[1] 2  
[1] 2.236068  
[1] 2.44949  
[1] 2.645751  
[1] 2.828427  
[1] 3  
[1] 3.162278
```

Q2.5a

```
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  
  
for (t in 2:50) {  
  N[t] = N[t - 1] * lambda # Apply the equation  
}  
  
# store data  
popn_data2 <- tibble(years = 1:years, # Make the years column = 1, 2, 3, ..., 20  
  popn = N)
```

Q2.5b

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



Q 2.6

Paige: Summarize/across makes the most sense, but probably has narrower applications than a for loop does. Having a strong understanding of a for loop has a lot of potential uses!

Leah: For loops make strong intuitive sense!

Q 2.7

```
# Store a vector of unique species names from the Species column of Iris  
spp_names <- unique(iris$Species)  
  
# Repeat the for loop for the number of unique species names. I.e., 3 species
```

```

# times
for (i in 1:length(spp_names)) {

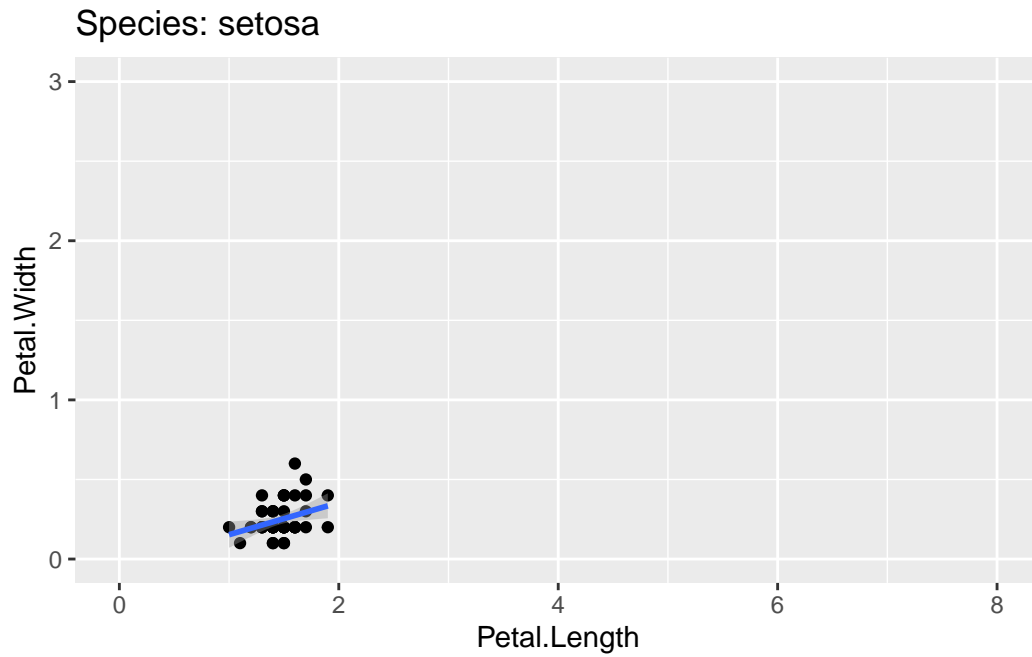
  filt_data <- iris %>%
    # for each loop, create a filter where only one of the species names present.
    #creates 3 filt_data tibbles, one for each species
    filter(Species == spp_names[i])

  # for each loop, plot just that filtered data
  plot <- filt_data %>%
    # map aesthetics to x and y axis using ggplot
    ggplot(aes(x = Petal.Length,
               y = Petal.Width)) +
    # scatter plot
    geom_point() +
    # with a line through it
    geom_smooth(method = "lm") +
    # set the x and y axis limits
    lims(x = c(0,8),
          y = c(0,3)) +
    # name the plot the species name that is specified for loop
    ggtitle(paste("Species:", spp_names[i]))

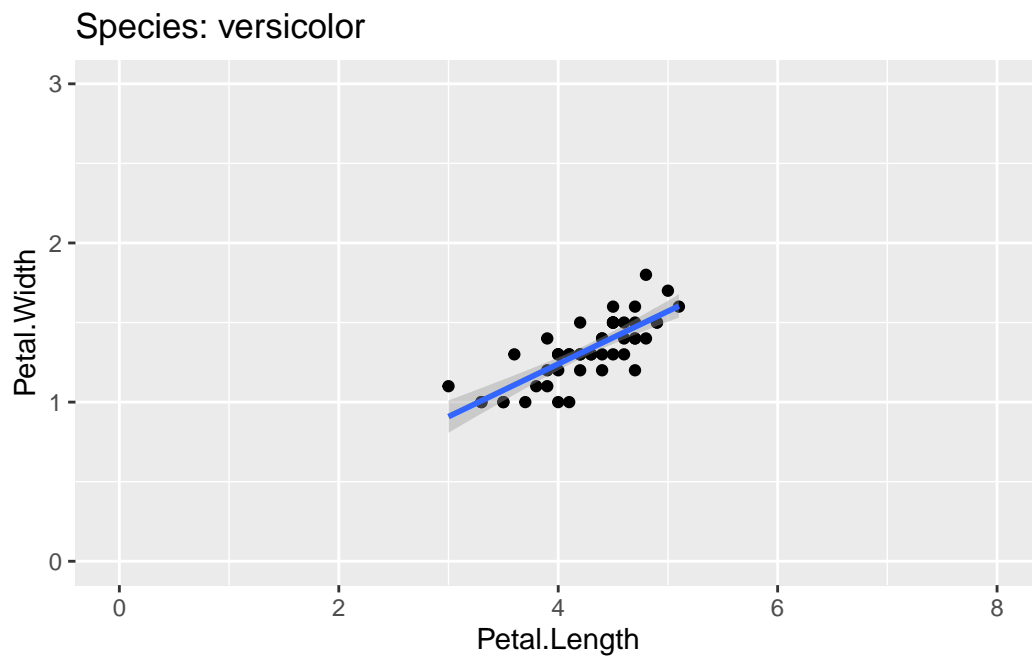
  # actually print/show the plot(s)
  print(plot)
}

```

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



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



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

Species: virginica

