2.2: Writing Functions

Learning Outcomes

- Students will be able to apply fundamental components of the biology of recirculating aquaculture systems to hypothesize how these components could lead to unhealthy fish.
- Students will be able to write custom functions that perform simple tasks.
- Students will be able to describe the utility of iteration.

Aquaculture, Immunology, and Disease

What is aquaculture anyway, and how might it contribute to disease? Here's a link to the presentation.

Data Exploration

Given the correlation we found earlier, it does seem like the fish are our likely culprit.

Since we've narrowed down the issue, we should ask our aquaculture specialists to provide us with some data about the tanks.

Group Brainstorm

Based on what we know about aquaculture systems, what types of data should we ask for to get to the bottom of this issue? Spend about 3 minutes brainstorming with your group and be ready to report back.

The Data We Have

```r
# Load the tidyverse
library(tidyverse)

# Read in the data
tank_data <- read_csv("data/fish_tank_data.csv")

# Look at the data
glimpse(tank_data)
```

```
Rows: 1,000
Columns: 7
$ tank_id        <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
$ species        <chr> "tilapia", "tilapia", "tilapia", "tilapia", "tilapia", ~
$ avg_daily_temp <dbl> 23.57285, 23.75280, 23.07280, 23.74601, 24.40522, 23.25~
$ num_fish       <dbl> 105, 102, 109, 98, 103, 97, 101, 99, 102, 95, 100, 102,~
$ day_length     <dbl> 10, 10, 10, 12, 10, 10, 11, 10, 9, 10, 10, 12, 11, 11, ~
$ tank_volume    <dbl> 400.7182, 399.4300, 399.4878, 400.8874, 399.8796, 399.8~
$ size_day_30    <dbl> 2779.726, 2785.846, 2781.342, 2784.785, 2786.340, 2782.~
```

Discussion of the problem

Trout are a cold-water species whereas tilapia are a warm-water species. After doing some reading, we've come up with some critical values for each species.

Water temperatures below 59º F for trout and below 75º F for tilapia are critically low temperatures and can result in suppression of the immune system.

Great, we now know what the critical temperature cutoffs are for both species — but there's a problem. The cut-off temperatures that we have are in Fahrenheit but the temperature values in our data frame are in Celsius.

Our goal with this data is to ascertain if any of the tanks are below the critical temperature. Before we can do this, though, we need to convert tank temperatures from Celsius to Fahrenheit. Luckily for us, we've already done this before! (Hint: remember the `mutate()` function?).

Now, we are going to run through a slightly different way to accomplish this task. It still involves `mutate()`, but we need to take some additional steps first before we use it.

Custom Functions

We've already used functions a lot in this class, and it turns out that we can write our own functions to perform specific tasks.

This has lot of varied uses and can be very powerful. The key takeaway here, though, is that we can write a function that converts Celsius to Fahrenheit and apply it to our entire data set in a few lines of code.

General Syntax Example

Making a function involves creating local variables that act as a stand-in for the values you want to use later on. In the case below, our local variables are `x`, `y`, and `z`. You can see that they are local because they only circulate within the function and are not being pulled in from outside. However, once we are done defining our function we can assign values to the input variables (in this case, `x` and `y`) that the function will do its magic with. You can see this in the example chunk below, where `x = 2` and `y = 4`. All manipulations done with the input variables are contained between the curly brackets (`{ }`), and of course, we signify the start a function with `function()`.

Function syntax follows the structure in the code chunk below, which can be viewed as:

`name_of_new_function <- function(input1, input2){`

`what do you want to do with your input variables? (add, subtract, etc.)`

`output <- input1 + input2`

`return(output)`

`}`

And we can see it in action here:

```
# General syntax
new_sum <- function(x,y){
  z <- x + y
  return(z)
}

# Examples when using the new function
new_sum(x = 2, y = 4)
```

```
[1] 6
```

```
new_sum(2, 4)
```

```
[1] 6
```

```
new_sum(7, 19)
```

```
[1] 26
```

Building Our Custom Function

In your groups, build a custom function called `c_to_f()` that converts a single value of Celsius to Fahrenheit. Remember, the equation is `Celsius = Fahrenheit * (9/5) + 32`. Instead of two arguments like we have above (`x`, `y`), this function will only require one input value (`c`).

Be prepared to show your code and run some test values.

```
c_to_f <- function(c = NULL){
  f <- (c * (9/5)) + 32
  return(f)
}

c_to_f(c = 40)
```

```
[1] 104
```

Thinking about Iteration

So, we have a great function that we can use to convert a Celsius value to Fahrenheit, but how do we apply that to *every* value in the temperature column, not just one? We've built the function for one value, not for a whole vector of values…

This is where `mutate()` comes in! As we've seen in the past, `mutate()` applies whatever function or mathematical formula we've given to it to each value in the column. It essentially is going row by row: applying the function or formula to one row, returning the new value in a new column, then moving to the next row to do the same.

```
# Let's first demonstrate applying our new function to a vector
c_to_f(tank_data$avg_daily_temp)
```

```
  [1] 74.43113 74.75503 73.53104 74.74281 75.92940 73.86383 74.31040 76.83799
  [9] 74.61511 74.87889 75.97796 75.12147 73.19913 73.29417 73.93870 74.66364
 [17] 74.83470 75.66626 73.70134 75.70409 76.46928 74.38196 75.24622 75.11160
 [25] 74.12701 74.80457 75.34147 74.86631 76.01130 75.44187 74.71827 75.01918
 [33] 76.51237 76.51730 73.27106 73.85835 74.27350 75.86871 73.20866 74.63055
 [41] 75.45388 73.82945 74.25658 74.59991 72.82957 75.27582 75.93890 74.83474
 [49] 74.41354 73.97254 75.02370 74.11599 76.44499 74.47102 75.26821 76.40062
 [57] 75.50111 75.18067 76.86133 75.24810 74.88691 75.63106 75.25761 73.64460
 [65] 75.89521 74.11780 75.15073 76.32487 74.95296 74.79653 76.51101 75.83362
 [73] 75.55690 75.14052 74.82020 75.08606 75.52449 76.95815 75.80831 73.10074
 [81] 75.02855 76.22341 75.64616 75.23696 75.17291 74.70379 74.43949 75.45942
 [89] 75.49829 74.67938 74.98023 73.80162 74.95824 74.66393 76.18906 74.98713
 [97] 74.83729 76.66777 75.90083 74.58555 76.30435 76.06749 75.01741 74.85544
[105] 74.43440 75.31092 74.50235 74.78281 74.95587 75.59466 74.66381 74.97921
[113] 73.58013 75.04360 73.21218 74.02326 73.64192 75.28364 74.45243 73.64938
[121] 75.93994 74.58181 76.25739 75.00178 74.22845 76.31685 74.41358 74.44657
[129] 74.83803 74.55838 74.66919 75.35266 75.08914 74.66152 74.05380 74.39329
[137] 76.60488 75.98699 74.62135 76.51778 73.53161 74.85214 75.05827 74.05406
[145] 74.46643 75.30342 75.05014 74.63223 75.44543 72.75165 73.48342 73.80455
[153] 73.50949 74.64755 75.00175 74.57649 74.11708 73.87874 75.27300 75.28975
[161] 74.98249 75.48612 76.41550 75.16506 74.98727 74.96898 74.74179 74.24449
[169] 75.27430 73.67490 75.65908 75.04681 76.79113 75.26938 74.79354 74.01127
[177] 74.76849 75.36104 73.74682 74.30743 75.67318 75.99384 75.66748 74.10130
[185] 76.20572 75.83375 74.37115 73.50744 73.73622 74.50636 73.27653 76.26933
[193] 75.71124 75.86047 75.73344 73.18854 74.28251 74.65673 74.65480 75.40234
[201] 73.87881 76.52272 75.22239 75.24770 75.48636 76.14890 76.46642 74.57311
[209] 74.99249 76.16651 75.82809 75.16704 73.94868 76.90079 74.84575 74.34058
[217] 75.61323 74.84086 75.29631 74.81182 75.49209 76.00486 76.05246 75.87944
[225] 74.79180 77.26703 73.58652 74.54782 76.88003 76.32634 74.80925 74.98680
[233] 73.75446 76.27250 75.27961 74.37445 74.76251 73.88835 74.85354 75.71155
[241] 73.60666 75.28570 76.18128 73.27245 74.02696 74.37579 74.73682 74.73227
[249] 73.37703 74.59072 74.23939 74.41314 74.90766 74.66029 75.12877 75.62673
[257] 76.32683 75.72618 74.85168 76.70432 76.28197 74.71700 75.30343 75.91907
[265] 75.24439 75.65435 75.19472 73.92471 73.60380 75.20492 75.66565 75.39755
[273] 75.25493 75.77523 75.66162 73.38134 74.86907 75.68101 73.85461 74.98361
[281] 75.46433 75.10175 76.63125 75.85701 75.17118 74.53800 75.81667 75.88907
[289] 74.43176 74.44114 74.48623 74.91623 74.28739 75.51695 75.94118 76.48571
[297] 76.14303 74.66342 74.97728 74.53391 73.52488 73.58812 76.11815 75.46769
[305] 75.19182 74.13072 76.31003 75.30696 75.02037 75.74875 75.73139 75.77816
[313] 74.34561 73.84045 75.24158 75.76953 73.13234 74.61631 74.81177 75.26779
[321] 74.23635 73.82120 76.31054 74.35616 76.07601 73.95509 74.05772 74.22910
[329] 75.43011 73.42268 76.79148 75.21833 74.80185 73.36899 73.14975 75.79115
[337] 74.61919 75.98506 73.42636 75.22286 73.99257 75.46905 73.89113 75.37155
```

```
[345] 74.05730 74.66263 74.66438 74.18706 74.49813 74.32892 74.23642 74.67756
[353] 75.04411 74.30414 74.70724 75.08641 75.71841 75.51808 74.93100 75.81793
[361] 74.78730 74.63517 74.97826 75.51747 74.52114 75.12071 78.13482 75.46619
[369] 74.46580 74.73910 75.67454 75.50536 75.52796 74.30964 76.21489 74.70730
[377] 76.88057 74.58457 74.43064 75.03877 74.36608 75.06382 75.98336 74.87293
[385] 74.22894 75.06329 74.27155 75.68821 74.24253 73.94639 73.49113 75.11725
[393] 75.85721 76.07618 75.18791 76.24006 75.62590 75.02484 74.42331 72.60027
[401] 73.52044 76.51052 74.28500 75.40433 73.90875 76.99178 76.35279 75.41119
[409] 74.13315 75.19382 75.50201 74.48111 75.90511 75.11015 75.25285 74.99648
[417] 74.47208 75.85869 74.68220 73.34307 75.04739 76.44738 75.31309 75.52913
[425] 76.26223 75.73930 76.33215 73.74962 73.33994 74.70288 74.19174 75.01108
[433] 74.63574 74.76168 74.78990 74.75733 73.00951 74.26312 76.02842 74.63533
[441] 75.14596 74.96193 74.11402 74.59574 76.09305 74.53926 74.74604 74.40459
[449] 77.27062 74.78709 74.36770 72.54340 75.09833 74.88731 73.50395 74.78807
[457] 75.05173 75.12390 74.90934 75.99915 74.33275 74.38525 74.10922 73.42221
[465] 76.64918 74.47975 75.12581 75.25714 75.35792 74.69494 75.62008 73.53747
[473] 75.71740 74.33907 75.53669 73.35504 74.45043 76.12163 76.03898 75.61239
[481] 74.80586 73.71626 73.48871 74.12853 73.84396 77.17008 76.43512 73.98444
[489] 76.04977 74.46583 76.01115 73.91292 75.92073 74.29250 74.63153 75.00484
[497] 74.52828 74.66825 75.29207 74.77686 76.19662 75.30999 75.69891 73.15895
[505] 73.85873 74.07277 75.41135 74.74135 74.26356 75.63118 75.28181 74.17680
[513] 74.68805 76.18352 74.81882 75.24053 75.04616 76.11425 75.36269 75.19598
[521] 74.37596 76.61887 73.65574 75.51871 74.48345 76.03400 75.57605 75.55502
[529] 75.65533 76.53720 75.03611 73.88219 74.41748 74.08832 76.36027 74.93291
[537] 75.66116 75.06712 74.36560 75.25447 75.49461 76.83779 75.88841 75.24546
[545] 73.47890 73.74035 74.00327 76.70369 74.94458 76.21075 74.29925 75.06979
[553] 76.01745 75.82861 75.54548 75.42884 73.98611 75.54789 76.09116 75.88274
[561] 76.46166 74.96008 75.43254 74.86664 74.29002 74.87314 74.85198 76.87914
[569] 74.06174 75.85227 75.67760 74.07233 75.48141 74.66367 74.93047 74.96150
[577] 74.49903 73.48692 73.68900 75.15058 75.11194 74.39278 74.14565 75.62441
[585] 74.94064 75.99647 74.85537 76.45970 75.82955 75.49968 75.89981 74.64502
[593] 74.74139 75.52825 75.47099 74.43561 75.00259 76.13283 75.72467 74.57316
[601] 75.02189 73.51909 75.54535 74.09301 73.49334 75.41178 74.95289 74.11120
[609] 74.07679 74.21184 74.83747 74.92889 74.04358 74.96736 74.76160 73.12468
[617] 75.29538 75.22185 75.34399 75.06791 76.37536 74.09531 73.46627 75.90204
[625] 76.25114 75.31286 77.02115 74.14008 75.03220 75.24348 73.99906 74.97442
[633] 75.66312 73.25093 75.45421 74.73571 73.21914 74.86915 75.43225 74.53833
[641] 75.61747 74.80382 76.44066 74.81603 76.04032 75.44944 74.26456 75.84708
[649] 74.68048 75.09555 73.92737 75.45199 74.56045 75.22312 75.03210 75.12464
[657] 75.52245 76.29832 74.50342 74.45557 74.70536 75.87068 74.94073 75.33082
[665] 74.72678 75.03815 75.78069 74.17345 75.36708 75.10770 74.96127 77.29781
[673] 76.23836 73.81312 74.40241 75.18181 74.61472 74.84399 74.73681 74.81869
[681] 75.35093 74.39910 73.61451 74.36612 73.33844 75.57199 73.31930 74.69243
```

```
[689] 75.09463 75.37939 76.62129 75.74483 75.38812 73.98151 75.04188 75.28739
[697] 75.18751 76.01130 74.61666 74.21020 75.79813 76.06483 75.31910 74.31577
[705] 74.67929 76.07411 73.80797 74.82510 73.71311 75.91244 75.04805 76.57705
[713] 75.72923 73.86751 75.28411 74.13344 75.53956 73.85239 73.03549 73.49739
[721] 76.82092 74.90142 74.57528 75.56377 75.34453 75.50219 76.00495 74.77230
[729] 75.33662 74.45439 74.17300 74.23845 73.32387 73.52121 76.13298 73.78813
[737] 75.41284 76.59846 75.35748 74.94102 74.53078 75.59105 74.86910 74.57358
[745] 75.53979 75.67722 74.04404 75.08682 74.43477 73.90495 58.71655 58.60596
[753] 59.81279 58.92998 58.74580 61.62783 58.89574 57.72173 58.00128 58.23492
[761] 57.46868 58.14439 58.33435 59.21383 58.26875 59.21490 59.93676 57.84037
[769] 58.93440 57.99766 58.40757 58.86463 58.19380 58.86269 58.96014 59.35988
[777] 58.73265 58.99126 58.88987 58.61232 59.43563 57.55058 58.64003 59.29693
[785] 59.85673 58.43464 58.53746 59.59014 57.99131 60.11249 59.61312 58.65781
[793] 58.48004 57.33794 59.90118 55.96181 58.86370 59.51051 59.25745 59.36329
[801] 60.21101 58.21819 60.41045 59.13623 57.49084 61.21628 58.54168 60.98868
[809] 59.25941 59.30273 58.45916 58.29524 59.36254 58.77167 59.30608 58.18048
[817] 58.51745 60.10645 57.31615 59.57920 59.00116 59.34116 58.70359 58.77237
[825] 58.91715 58.88182 59.58683 59.02189 60.04784 59.61932 59.35197 60.69102
[833] 59.12942 59.34041 59.78739 58.66497 60.09224 58.85611 59.29852 60.08068
[841] 59.05809 58.44424 59.68582 59.47273 59.26000 59.36474 60.53905 59.03931
[849] 58.38699 59.14683 60.18130 57.86961 58.60957 58.99026 59.13557 58.51197
[857] 59.14811 59.47670 57.76631 58.73476 59.12640 60.01568 58.48686 57.62616
[865] 59.13953 58.66865 58.50021 59.29785 57.51375 61.24982 59.90547 58.47528
[873] 57.63368 58.72005 58.95607 58.73648 60.54165 57.87001 59.11123 59.45932
[881] 61.56960 58.72222 58.80271 58.80800 58.79589 59.49097 59.32121 58.48496
[889] 58.10383 58.12429 59.86692 59.20175 59.21730 58.58847 60.27672 58.10896
[897] 58.29106 59.94312 59.80183 58.42179 59.85023 58.97875 59.70343 59.56826
[905] 58.55075 58.60945 59.08787 58.03444 57.76502 59.16565 57.11930 58.16243
[913] 60.70687 58.89522 58.23583 60.20542 58.14132 58.27318 59.95025 57.69363
[921] 58.26243 59.61055 58.53237 58.24523 59.25394 59.63806 57.57931 58.95890
[929] 58.91506 60.05179 60.07972 58.30900 58.69114 60.87270 59.38890 60.01367
[937] 56.88199 57.53978 58.52811 59.30322 60.71852 60.31373 57.77659 59.33987
[945] 58.45096 58.57272 59.70338 60.33623 58.94100 59.03894 59.13207 59.44900
[953] 59.19860 58.80944 58.32800 58.39747 60.27306 59.47031 58.28749 57.83506
[961] 60.66958 58.08557 58.57839 58.88083 58.54565 58.98934 59.74796 59.64448
[969] 57.56594 59.74780 57.28264 58.39450 60.28522 59.68049 58.99770 57.25823
[977] 58.25759 59.30419 59.18225 57.00076 59.96050 57.83463 59.84659 58.17613
[985] 59.47178 59.16409 59.76432 58.55604 58.55329 58.18867 58.57248 60.31412
[993] 59.65095 58.74366 60.13083 60.32541 58.37594 60.20369 60.00230 59.09448
```

```r
# Mutate is basically doing the same thing
tank_data <- tank_data %>%
  mutate(avg_daily_temp_F = c_to_f(avg_daily_temp))

# View mutate() results
tank_data
```

```
# A tibble: 1,000 x 8
   tank_id species avg_daily_temp num_fish day_length tank_volume size_day_30
     <dbl> <chr>            <dbl>    <dbl>      <dbl>       <dbl>       <dbl>
 1       1 tilapia           23.6      105         10        401.       2780.
 2       2 tilapia           23.8      102         10        399.       2786.
 3       3 tilapia           23.1      109         10        399.       2781.
 4       4 tilapia           23.7       98         12        401.       2785.
 5       5 tilapia           24.4      103         10        400.       2786.
 6       6 tilapia           23.3       97         10        400.       2783.
 7       7 tilapia           23.5      101         11        400.       2784.
 8       8 tilapia           24.9       99         10        401.       2789.
 9       9 tilapia           23.7      102          9        400.       2788.
10      10 tilapia           23.8       95         10        399.       2785.
# i 990 more rows
# i 1 more variable: avg_daily_temp_F <dbl>
```

Why Functions?

We can easily fit the equation for converting from Celsius to Fahrenheit into a mutate call. We certainly didn't need to write our own function to do it. So why did we learn how to do that?

Main Ways to Iterate

There are many different approaches to iterating in R, especially because we are typically working with vectorized data (columns in data frames). We will talk about the other options down the road. Today, we used #3, the `mutate()` function from the `tidyverse`.

1. `for` loops
2. `apply`/`map` functions
3. using `dplyr` (`tidyverse`) functions

Filtering Our Data

Thankfully, we've solved our first problem! However, we haven't yet answered the original question.

Are any tanks are below the temperature cutoffs for each species? If so, how many?

Let's tackle that question:

```
# count() is a new function for most of us
# It does what it sounds like -- count the number of results

# Count number of tanks below temperature cutoff for TILAPIA
tank_data %>%
  filter(species == "tilapia" & avg_daily_temp_F < 75) %>%
  count()
```

```
# A tibble: 1 x 1
      n
  <int>
1   383
```

```
# for TROUT
tank_data %>%
  filter(species == "trout" & avg_daily_temp_F < 59) %>%
  count()
```

```
# A tibble: 1 x 1
      n
  <int>
1   131
```

```
# Let's introduce some additional syntax: "&" and "|"

# Now we can get counts for both fish species in one go!
tank_data %>%
  filter(species == "tilapia" & avg_daily_temp_F < 75 |
         species == "trout" & avg_daily_temp_F < 59) %>%
  group_by(species) %>%
  summarize(n = n()) # Another way to get the count is summarize() with n()
```

```
# A tibble: 2 x 2
  species     n
  <chr>   <int>
1 tilapia   383
2 trout     131
```