Loops and Functions Homework

Katie Temple

2025-03-27

Table of Contents

# Loops, Functions, and Iterations

library (ggplot2)  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ lubridate 1.9.4 ✔ tibble 3.2.1  
## ✔ purrr 1.0.2 ✔ tidyr 1.3.1  
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library("drc")

## Loading required package: MASS  
##   
## Attaching package: 'MASS'  
##   
## The following object is masked from 'package:dplyr':  
##   
## select  
##   
##   
## 'drc' has been loaded.  
##   
## Please cite R and 'drc' if used for a publication,  
## for references type 'citation()' and 'citation('drc')'.  
##   
##   
## Attaching package: 'drc'  
##   
## The following objects are masked from 'package:stats':  
##   
## gaussian, getInitial

## Functions

“Things that do things to other things”

You can make a function so that you don’t have to handwrite the equation each time increasing the chance of an error and creating non-reproducible work. It also makes your life easier.

#Example: Converting Fahrenheit to Celcius  
  
#here's the equation by hand  
#(5\*(degree\_f-32)/9) degree\_f= the degrees in f you want to convert to c  
(5\*(32-32)/9)

## [1] 0

#0  
  
#another example for today's forecast   
(5\*(74-32)/9)

## [1] 23.33333

#23.33  
  
#rather than having to write out the equation by hand and risking mistakes, you   
#can covert the equation into a function.  
F\_to\_C <- function(farenheit\_temp){ #farenheit\_temp is what you will input each time  
 celsius <- (5\*(farenheit\_temp - 32)/9) #this defines what the function is actually doing which is the rquation for converting f to c. Here you can see that farenheit\_temp will be put in the equation and that's for you to enter each time/what you want r to calculate  
 return (celsius) #return is what the result of the function is which is returning the value calculated for celcius  
} #the function will appear in your global environment.  
#when you see the function in the environment you can see what value/variable you need to input into the function  
F\_to\_C(32)

## [1] 0

#0 which matches what we calculated by hand  
  
F\_to\_C(74)

## [1] 23.33333

#23.33

### Anatomy of a function

The below text is directly copied from the assignment page.

We first start with naming the function something using the backwards arrow. Then we type “function()”.

After opening a new function we type the curly brackets. We will type the code we want to perform within the curly brackets.

Inside the parentheses after function is where we put our variables separated by commas.

“sample.function <- function(… variable goes here …){ …. code goes here…. return(… output …) }”

1. So, in the example above we want to put the variable inside the parentheses. This is what we are going to input into the function.
2. Then we input what we want to do to the variable inside the brackets
3. Then all functions need a return() - this is what we want the output of the function to be

Make sense?

#creating own function with example of c to f  
#F = C x (9/5)+32  
c\_to\_f <- function (celcius\_temp){  
 fahrenheit <- ((celcius\_temp\*(9/5)) + 32)  
 return (fahrenheit)  
}  
c\_to\_f(0)

## [1] 32

#returns 32  
  
c\_to\_f(23.33)

## [1] 73.994

#returns 73.94 which rounded is 74

## Iterations

“Iterations are something you do over and over again” Iterations not only prevent copy and paste errors, they also help with sanity checks since it’s something you do over and over and over through a data simulation.

### The rep() function

Repeats things so you don’t have to

rep("A", 3) #repeats A three times

## [1] "A" "A" "A"

#[1] "A" "A" "A"  
rep(c("A", "B"), 5) #repeats A B five times

## [1] "A" "B" "A" "B" "A" "B" "A" "B" "A" "B"

# [1] "A" "B" "A" "B" "A" "B" "A" "B" "A" "B"  
  
rep(c(1,2,3,4), times = 4) #repeats "1,2,3,4," 4 times

## [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

#[1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4  
  
rep(c(1,2,5,2), times = 4, each = 4) #repeats 1 four times, 2 four times, 5 four times, and 2 four times.

## [1] 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2 1 1 1 1 2 2  
## [39] 2 2 5 5 5 5 2 2 2 2 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2

#[1] 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2 1 1 1  
#[36] 1 2 2 2 2 5 5 5 5 2 2 2 2 1 1 1 1 2 2 2 2 5 5 5 5 2 2 2 2

### The seq() function

The seq() (sequence) command allows you to write *sequences* of numbers easily.

seq(from = 1, to = 7) #sequnce of numbers from 1 to 7

## [1] 1 2 3 4 5 6 7

#[1] 1 2 3 4 5 6 7  
  
seq(from = 0, to = 10, by = 2) #squence from 1 to 10 counting by 2s

## [1] 0 2 4 6 8 10

#[1] 0 2 4 6 8 10  
  
#combining seq() and rep()  
rep(seq (from = 0, to = 10, by = 2), times = 3, each = 2)

## [1] 0 0 2 2 4 4 6 6 8 8 10 10 0 0 2 2 4 4 6 6 8 8 10 10 0  
## [26] 0 2 2 4 4 6 6 8 8 10 10

#[1] 0 0 2 2 4 4 6 6 8 8 10 10 0 0 2 2 4 4 6 6 8 8 10  
#[24] 10 0 0 2 2 4 4 6 6 8 8 10 10  
#sequence from 0 to 10 counted by 2s repeats each number generated 3 times and each number in the repeated 2 each

### seq\_along()

This function “seq\_along()” allows you to generate a *sequence* of numbers on *non-integer* values. This will become useful later on when we want to loop elements within a dataframe.

#use the built in LETTERS vector as an example  
LETTERS

## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"  
## [20] "T" "U" "V" "W" "X" "Y" "Z"

#[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q"  
#[18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"  
seq\_along(LETTERS[1:5]) #returns the numbers 1-5 not the actual letters

## [1] 1 2 3 4 5

#[1] 1 2 3 4 5

## The for loop

The for loop has as certain algorithm no matter what language your coding in. You tell the code how many iterations you want looped.

There is an accompanying video that goes more in depth of the anatomy of a for loop

for (i in 1:10) { #normal syntax is i denoting subscript i. Include the sequence you want here.  
 print(i\*2) #the next line is where you type code. What this is saying is fr each value of i it will iterate 1-10 and substitute the value for i and multiple it by 2( 1\*2, 2\*2, 3\*2...)  
}

## [1] 2  
## [1] 4  
## [1] 6  
## [1] 8  
## [1] 10  
## [1] 12  
## [1] 14  
## [1] 16  
## [1] 18  
## [1] 20

#The print function was used because you have to be very explicit in for loops.  
#[1] 2  
#[1] 4  
#[1] 6  
#[1] 8  
#[1] 10  
#[1] 12  
#[1] 14  
#[1] 16  
#[1] 18  
#[1] 20

### A more complicated example

We can create a for loop that uses our function of f to c for for values -30:100

for (i in -30:100){ #for every i/value in the sequence of -30 to 100. This is your loop  
 result <- F\_to\_C(i) #convert Fahrenheit temps to Celsius for the value i (in the sequence) and name the product result  
 print(result) #displau result  
}

## [1] -34.44444  
## [1] -33.88889  
## [1] -33.33333  
## [1] -32.77778  
## [1] -32.22222  
## [1] -31.66667  
## [1] -31.11111  
## [1] -30.55556  
## [1] -30  
## [1] -29.44444  
## [1] -28.88889  
## [1] -28.33333  
## [1] -27.77778  
## [1] -27.22222  
## [1] -26.66667  
## [1] -26.11111  
## [1] -25.55556  
## [1] -25  
## [1] -24.44444  
## [1] -23.88889  
## [1] -23.33333  
## [1] -22.77778  
## [1] -22.22222  
## [1] -21.66667  
## [1] -21.11111  
## [1] -20.55556  
## [1] -20  
## [1] -19.44444  
## [1] -18.88889  
## [1] -18.33333  
## [1] -17.77778  
## [1] -17.22222  
## [1] -16.66667  
## [1] -16.11111  
## [1] -15.55556  
## [1] -15  
## [1] -14.44444  
## [1] -13.88889  
## [1] -13.33333  
## [1] -12.77778  
## [1] -12.22222  
## [1] -11.66667  
## [1] -11.11111  
## [1] -10.55556  
## [1] -10  
## [1] -9.444444  
## [1] -8.888889  
## [1] -8.333333  
## [1] -7.777778  
## [1] -7.222222  
## [1] -6.666667  
## [1] -6.111111  
## [1] -5.555556  
## [1] -5  
## [1] -4.444444  
## [1] -3.888889  
## [1] -3.333333  
## [1] -2.777778  
## [1] -2.222222  
## [1] -1.666667  
## [1] -1.111111  
## [1] -0.5555556  
## [1] 0  
## [1] 0.5555556  
## [1] 1.111111  
## [1] 1.666667  
## [1] 2.222222  
## [1] 2.777778  
## [1] 3.333333  
## [1] 3.888889  
## [1] 4.444444  
## [1] 5  
## [1] 5.555556  
## [1] 6.111111  
## [1] 6.666667  
## [1] 7.222222  
## [1] 7.777778  
## [1] 8.333333  
## [1] 8.888889  
## [1] 9.444444  
## [1] 10  
## [1] 10.55556  
## [1] 11.11111  
## [1] 11.66667  
## [1] 12.22222  
## [1] 12.77778  
## [1] 13.33333  
## [1] 13.88889  
## [1] 14.44444  
## [1] 15  
## [1] 15.55556  
## [1] 16.11111  
## [1] 16.66667  
## [1] 17.22222  
## [1] 17.77778  
## [1] 18.33333  
## [1] 18.88889  
## [1] 19.44444  
## [1] 20  
## [1] 20.55556  
## [1] 21.11111  
## [1] 21.66667  
## [1] 22.22222  
## [1] 22.77778  
## [1] 23.33333  
## [1] 23.88889  
## [1] 24.44444  
## [1] 25  
## [1] 25.55556  
## [1] 26.11111  
## [1] 26.66667  
## [1] 27.22222  
## [1] 27.77778  
## [1] 28.33333  
## [1] 28.88889  
## [1] 29.44444  
## [1] 30  
## [1] 30.55556  
## [1] 31.11111  
## [1] 31.66667  
## [1] 32.22222  
## [1] 32.77778  
## [1] 33.33333  
## [1] 33.88889  
## [1] 34.44444  
## [1] 35  
## [1] 35.55556  
## [1] 36.11111  
## [1] 36.66667  
## [1] 37.22222  
## [1] 37.77778

#This only prints in the console area. You can do more with the result of the iterations.

If you want to actually do something with the results of your for loop you can.

#Example  
#1. Set a R object to NULL  
#2. Set your for loop  
#3. Save the result of your for loop into a dataframe each iteration  
#4. append one row of the dataframe to the null object each iteration of the loop.   
  
celcius.df <- NULL #null is empty object but want this variable in the future  
for (i in -30:100){  
 result\_i <- data.frame (F\_to\_C(i), i) #we want each iteration to be a 1 row dataframe. We can turn the result of our function into a dataframe name result. It will be two columns with our results of the iterations and what the iteration/i was was called i (for now)  
 celcius.df <- rbind.data.frame(celcius.df, result\_i)  
}  
#On the first iteration -30 will be plugged wherever there is i and will create a one row dataframe called result. Next the result dataframe is going to rowbind (rbind). This creates a new row and binds it to the celcius dataframe. It will do this for the values you've told it in the first line.  
#the result will be a celcius.df with all the values and then can be used to do what you want like plotting.  
View(celcius.df) #look at the dataframe