Coding Challenge 6: Loops and Functions

Katie Temple

2025-03-27

Table of Contents

## Question 1

2 pts. Regarding reproducibility, what is the main point of writing your own functions and iterations?

### Answer 1

The main point in writing your own functions is to reduce potential error through typos, copy and paste mistakes, , missremembering a formula, and that a lot of it is easier than doing by hand. Additionally, it clearly lays out why you’re doing what you’re doing and those trying to reproduce your analysis should be able to plug in the same values using the function. The same can be said for when needing have/write iterative values of something.

## Question 2

2 pts. In your own words, describe how to write a function and a for loop in R and how they work. Give me specifics like syntax, where to write code, and how the results are returned.

### Answer 2

When writing your own function by hand, you need to first know what you want to use the function for. In the example in the homework, we wanted to convert Fahrenheit temperatures to Celsius and used the standard conversion equation to do so.

#### Functions Answer

#Name your function and tell r you want to make a function.  
# Example: converting ounces to grams: The equation to do that is ounces\*28.35.   
#The first line of code is also where you want to name your input/what you want to manipulate   
#(in this case we want to convert ounces.)  
ounces\_to\_grams <- function (ounces){ #name your function and then tell r you want to make a function with "  
 #function() and inside the parenthesis put what you want to manipulate. Then use the squiggly   
 #brackets and put enter so that they encapsulate the next steps.  
 grams <- (ounces\*28.35) #write the equation by name with your variable that you're   
 #converting. In this case ounces. The output is name of the converted variable you want returned.  
 return(grams) #here you're saying that you want the value of the converted variable to grams returned  
} #close the function. This will show your Global Environment under the functions aspect.  
  
#now use your function. Convert some ounces to grams  
ounces\_to\_grams(16) #in this case you're converting 16 ounces to grams

## [1] 453.6

#[1] 453.6  
#Now check  
16\*28.35

## [1] 453.6

#[1] 453.6

#### For loops Answer

When writing for loops, you need to be explicit. For loops will loop pack to (in this example) each number in the sequence and manipulate it the way you wanted (in the example below multiplied by the ounces to gram conversion of 28.35). It will loop back until it has addressed each part of the sequence you’ve given. Each line generates a single row dataframe of the output unless you bind in (such as with rbind).

for(i in 0:10) {#i is what you want to iterate. Then do squiggle bracket to start and enter what you want to do  
 print (i\*28.35) #have i and say what you want to happen to i. In this case i want to multiply it by 28.35  
 #the print command is used to so that you can see the output. You have to be explicit with loops such as saying you want the output to be displayed. This is where you put your function (if you want your i and product of i in regard to function)  
}

## [1] 0  
## [1] 28.35  
## [1] 56.7  
## [1] 85.05  
## [1] 113.4  
## [1] 141.75  
## [1] 170.1  
## [1] 198.45  
## [1] 226.8  
## [1] 255.15  
## [1] 283.5

#[1] 0  
#[1] 28.35  
#[1] 56.7  
#[1] 85.05  
#[1] 113.4  
#[1] 141.75  
#[1] 170.1  
#[1] 198.45  
#[1] 226.8  
#[1] 255.15  
#[1] 283.5  
  
#Within the squiggle brackets, you can also do things like create dataframes. A way to think about it   
#is each line should be what you want to happen and then the next is what you want to happen next in regard to the previous line.

## Question 3

2 pts. Read in the Cities.csv file from Canvas using a relative file path.

cities <- read.csv("Cities.csv", na.strings = "na")  
str(cities)

## 'data.frame': 40 obs. of 10 variables:  
## $ city : chr "New York" "Los Angeles" "Chicago" "Miami" ...  
## $ city\_ascii : chr "New York" "Los Angeles" "Chicago" "Miami" ...  
## $ state\_id : chr "NY" "CA" "IL" "FL" ...  
## $ state\_name : chr "New York" "California" "Illinois" "Florida" ...  
## $ county\_fips: int 36081 6037 17031 12086 48201 48113 42101 13121 11001 25025 ...  
## $ county\_name: chr "Queens" "Los Angeles" "Cook" "Miami-Dade" ...  
## $ lat : num 40.7 34.1 41.8 25.8 29.8 ...  
## $ long : num -73.9 -118.4 -87.7 -80.2 -95.4 ...  
## $ population : int 18832416 11885717 8489066 6113982 6046392 5843632 5696588 5211164 5146120 4355184 ...  
## $ density : num 10944 3166 4590 4791 1386 ...

## Question 4

6 pts. Write a function to calculate the distance between two pairs of coordinates based on the Haversine formula (see below). The input into the function should be lat1, lon1, lat2, and lon2. The function should return the object distance\_km. All the code included needs to go into the function.

#Write a function using the steps in the assignment.  
# convert to radians  
distance <-function (lat1, lon1, lat2, lon2){  
# convert to radians  
rad.lat1 <- lat1 \* pi/180  
rad.lon1 <- lon1 \* pi/180  
rad.lat2 <- lat2 \* pi/180  
rad.lon2 <- lon2 \* pi/180  
  
# Haversine formula  
delta\_lat <- rad.lat2 - rad.lat1  
delta\_lon <- rad.lon2 - rad.lon1  
a <- sin(delta\_lat / 2)^2 + cos(rad.lat1) \* cos(rad.lat2) \* sin(delta\_lon / 2)^2  
c <- 2 \* asin(sqrt(a))   
  
# Earth's radius in kilometers  
earth\_radius <- 6378137  
  
# Calculate the distance  
distance\_km <- (earth\_radius \* c)/1000  
return (distance\_km)  
}

## Question 5

5 pts. Using your function, compute the distance between Auburn, AL and New York City

1. Subset/filter the Cities.csv data to include only the latitude and longitude values you need and input as input to your function.

b.The output of your function should be 1367.854 km

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  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.4 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.2   
## ── 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

str(cities)

## 'data.frame': 40 obs. of 10 variables:  
## $ city : chr "New York" "Los Angeles" "Chicago" "Miami" ...  
## $ city\_ascii : chr "New York" "Los Angeles" "Chicago" "Miami" ...  
## $ state\_id : chr "NY" "CA" "IL" "FL" ...  
## $ state\_name : chr "New York" "California" "Illinois" "Florida" ...  
## $ county\_fips: int 36081 6037 17031 12086 48201 48113 42101 13121 11001 25025 ...  
## $ county\_name: chr "Queens" "Los Angeles" "Cook" "Miami-Dade" ...  
## $ lat : num 40.7 34.1 41.8 25.8 29.8 ...  
## $ long : num -73.9 -118.4 -87.7 -80.2 -95.4 ...  
## $ population : int 18832416 11885717 8489066 6113982 6046392 5843632 5696588 5211164 5146120 4355184 ...  
## $ density : num 10944 3166 4590 4791 1386 ...

View(cities)  
latau=cities$lat[cities$city=="Auburn"]  
lonau=cities$long[cities$city=="Auburn"]  
latny=cities$lat[cities$city=="New York"]  
lonny=cities$long[cities$city=="New York"]  
  
distance(latau,lonau, latny, lonny)

## [1] 1367.854

#[1] 1367.854

## Question 6

6 pts. Now, use your function within a for loop to calculate the distance between all other cities in the data. The output of the first 9 iterations is shown below.

#need to change the function so that   
nm<-unique(cities$city)  
  
distance2 <-function (lat1, lon1, lat2, lon2){  
lat1=cities$lat[cities$city=="Auburn"]  
lon1=cities$long[cities$city=="Auburn"]  
lat2=cities$lat[cities$city==nm[[i]]]  
lon2=cities$lon[cities$city== nm[[i]]]  
 # convert to radians  
rad.lat1 <- lat1 \* pi/180  
rad.lon1 <- lon1 \* pi/180  
rad.lat2 <- lat2 \* pi/180  
rad.lon2 <- lon2 \* pi/180  
  
# Haversine formula  
delta\_lat <- rad.lat2 - rad.lat1  
delta\_lon <- rad.lon2 - rad.lon1  
a <- sin(delta\_lat / 2)^2 + cos(rad.lat1) \* cos(rad.lat2) \* sin(delta\_lon / 2)^2  
c <- 2 \* asin(sqrt(a))   
  
# Earth's radius in kilometers  
earth\_radius <- 6378137  
  
# Calculate the distance  
distance\_km <- (earth\_radius \* c)/1000  
return (distance\_km)  
}  
  
distances.df <-NULL  
for(i in seq\_along(nm)){  
 result\_i<- data.frame(distance2())  
 distances.df<-rbind.data.frame(distances.df, result\_i)  
}  
print(distances.df)

## distance2..  
## 1 1367.8540  
## 2 3051.8382  
## 3 1045.5213  
## 4 916.4138  
## 5 993.0298  
## 6 1056.0217  
## 7 1239.9732  
## 8 162.5121  
## 9 1036.9900  
## 10 1665.6985  
## 11 2476.2552  
## 12 1108.2288  
## 13 3507.9589  
## 14 3388.3656  
## 15 2951.3816  
## 16 1530.2000  
## 17 591.1181  
## 18 1363.2072  
## 19 1909.7897  
## 20 1380.1382  
## 21 2961.1199  
## 22 2752.8142  
## 23 1092.2595  
## 24 796.7541  
## 25 3479.5376  
## 26 1290.5492  
## 27 3301.9923  
## 28 1191.6657  
## 29 608.2035  
## 30 2504.6312  
## 31 3337.2781  
## 32 800.1452  
## 33 1001.0879  
## 34 732.5906  
## 35 1371.1633  
## 36 1091.8970  
## 37 1043.2727  
## 38 851.3423  
## 39 1382.3721  
## 40 0.0000

##Question 7 link: [link to GitHub](https://github.com/temkat/TempleReproducibilityClass2025/tree/5bdfb88dca4e0da5e664e870b14e8216a988b9af/LoopsFunctions/CodingChallenge6)