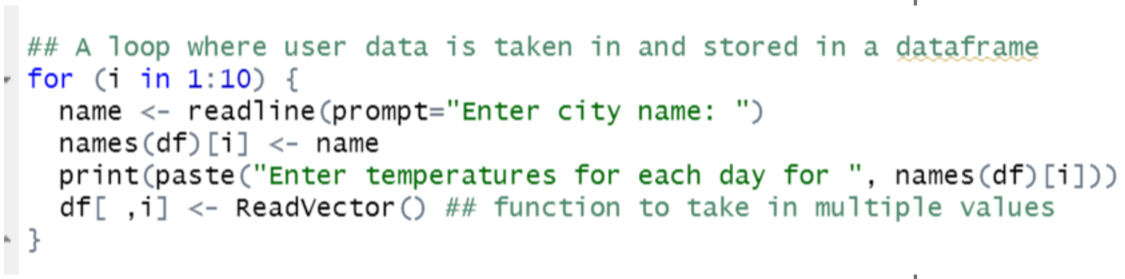


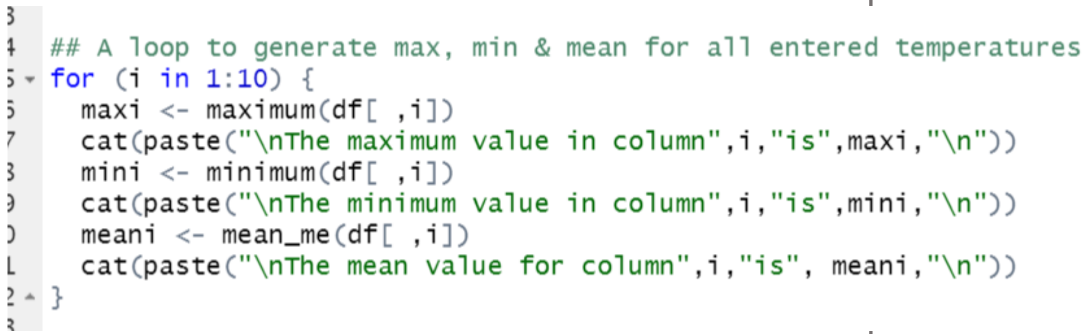
```
## A loop where user data is taken in and stored in a dataframe
for (i in 1:10) {
  name <- readline(prompt="Enter city name: ")
  names(df)[i] <- name
  print(paste("Enter temperatures for each day for ", names(df)[i]))
  df[ ,i] <- ReadVector() ## function to take in multiple values
}
```

Input User Data:

I used another ‘for’ loop to take input data from the user. I know this isn’t a very ‘user friendly’ way to generate data, but for the purposes of this exercise it works.

The user enters the city name first which is added to the first vector with the ‘names’ function. The ‘ReadVector’ algorithm takes in each temperature value, one at a time, and fills the vector.

The for loop then moves on to the next city, and so on until all ten cities’ values have been entered.

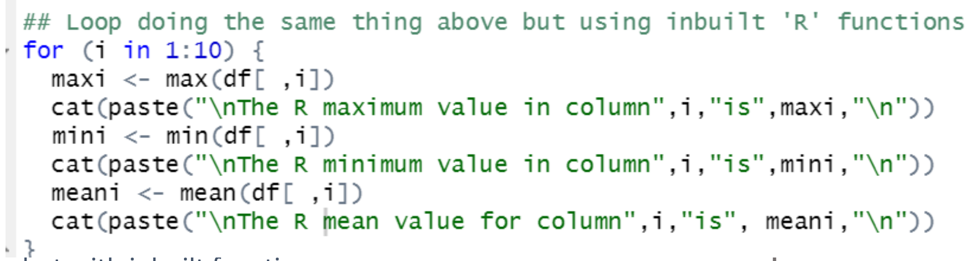


```
## A loop to generate max, min & mean for all entered temperatures
for (i in 1:10) {
  maxi <- maximum(df[ ,i])
  cat(paste("\nThe maximum value in column",i,"is",maxi,"\n"))
  mini <- minimum(df[ ,i])
  cat(paste("\nThe minimum value in column",i,"is",mini,"\n"))
  meani <- mean_me(df[ ,i])
  cat(paste("\nThe mean value for column",i,"is", meani,"\n"))
}
```

Print out max, min & mean:

Another ‘for’ loop (bit of a one-trick-pony I am) is used to generate min, max and mean values for each of the cities entered by the user. The loop takes the column number and then prints out the values for it (including the city’s name) to the screen.

After printing the three values for the first city it moves onto the second until all ten have been printed to screen.



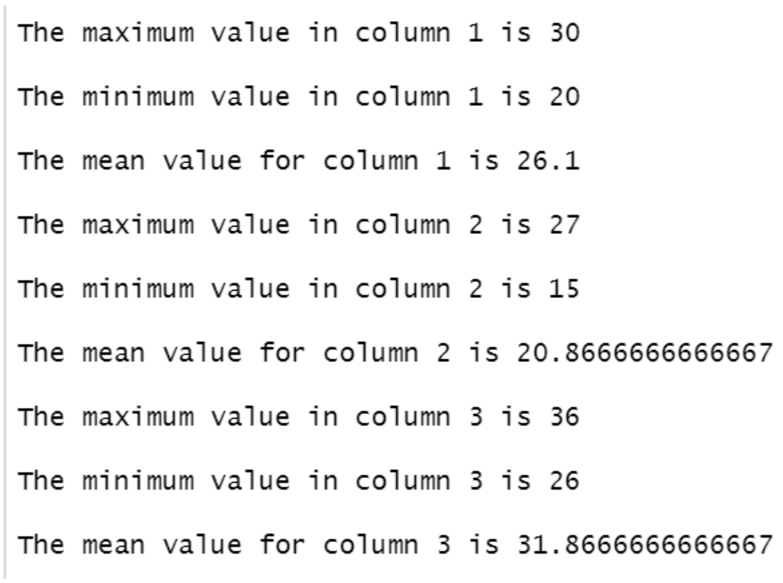
```
## Loop doing the same thing above but using inbuilt 'R' functions
for (i in 1:10) {
  maxi <- max(df[ ,i])
  cat(paste("\nThe R maximum value in column",i,"is",maxi,"\n"))
  mini <- min(df[ ,i])
  cat(paste("\nThe R minimum value in column",i,"is",mini,"\n"))
  meani <- mean(df[ ,i])
  cat(paste("\nThe R mean value for column",i,"is", meani,"\n"))
}
```

Same results but with inbuilt functions:

I repeat the same process as the previous but generate results with the inbuild R functions instead of my own.

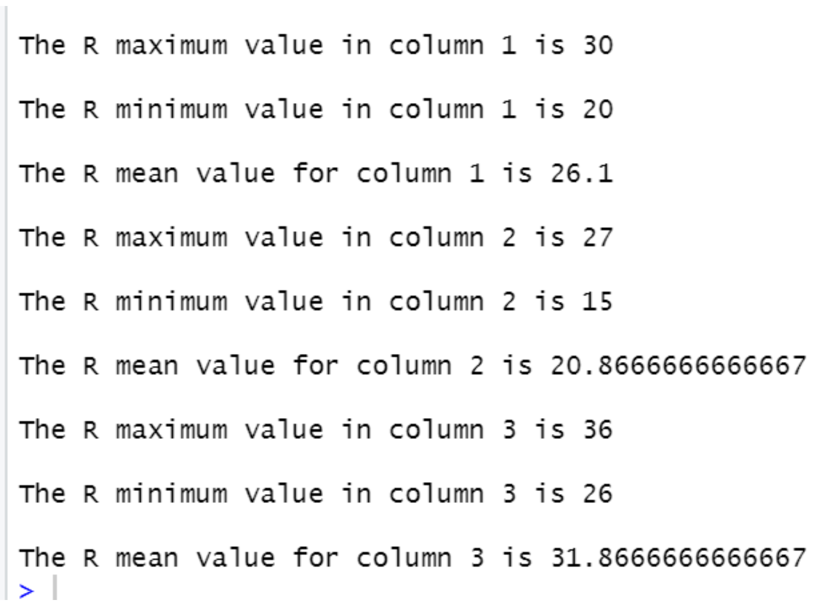
Results of the program using a second test program:

This is a print out of all the values that came from my own functions:



```
The maximum value in column 1 is 30

The minimum value in column 1 is 20

The mean value for column 1 is 26.1

The maximum value in column 2 is 27

The minimum value in column 2 is 15

The mean value for column 2 is 20.8666666666667

The maximum value in column 3 is 36

The minimum value in column 3 is 26

The mean value for column 3 is 31.8666666666667
```

This is a printout of the values generated form the R functions:



```
The R maximum value in column 1 is 30

The R minimum value in column 1 is 20

The R mean value for column 1 is 26.1

The R maximum value in column 2 is 27

The R minimum value in column 2 is 15

The R mean value for column 2 is 20.8666666666667

The R maximum value in column 3 is 36

The R minimum value in column 3 is 26

The R mean value for column 3 is 31.8666666666667
>
```

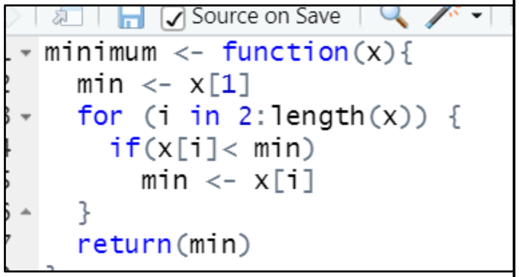
Happily, the results were the same.

1. Basic algorithmic thinking in R:

I started by designing each of the functions needed to run the program. It was stressed that it should be our focus, so I wanted to get it right.

Minimum function:

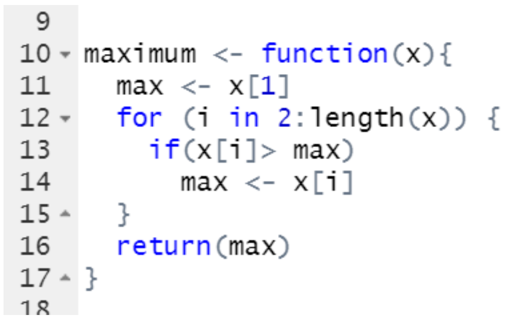
The function accepts a vector and assigns the first value in the list to ‘min’. It then cycles the rest of the values through a ‘for’ loop, from the second to the last in the list (length giving the function the point at which to stop), replacing the min value if a smaller one is found. This value is then returned to the parent program.



```r
minimum <- function(x){
  min <- x[1]
  for (i in 2:length(x)) {
    if(x[i]< min)
      min <- x[i]
  }
  return(min)
```

Maximum function:

The maximum function applies almost the same process as the minimum except, naturally, replacing the ‘less than’ sign with the ‘greater than’. It cycles through all supplied values and replaces ‘max’ with an element that is larger than it.

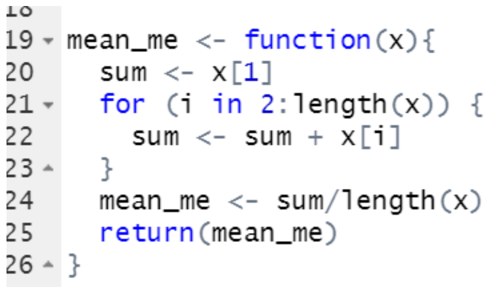


```r
maximum <- function(x){
  max <- x[1]
  for (i in 2:length(x)) {
    if(x[i]> max)
      max <- x[i]
  }
  return(max)
}
```

Mean function:

The mean function – which I christened ‘mean-me’ to avoid confusion – uses the same structure as the previous two functions (setting an initial value and then cycling through the rest with a for loop) but has an extra stage.

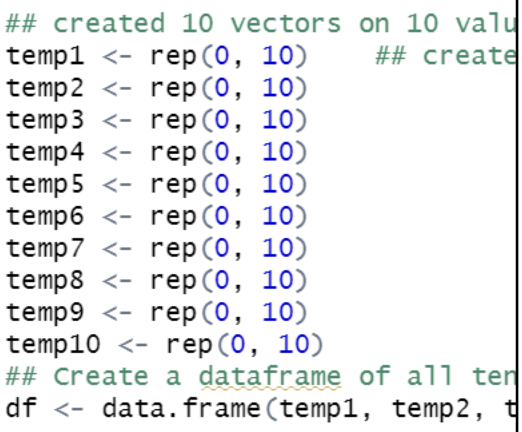
Sum becomes that number plus the next on the list. After it has collated all the numbers in the list it gets divided by the vector length. This gives the mean value for the entire vector.



```r
mean_me <- function(x){
  sum <- x[1]
  for (i in 2:length(x)) {
    sum <- sum + x[i]
  }
  mean_me <- sum/length(x)
  return(mean_me)
}
```

Dataframe creation:

No doubt far from the most efficient way to create a dataframe, but I created each vector individually and filled them with zeros. I did this because my program asks for the city name first and I needed values in the vector for the name to ‘stick’. All the vectors were placed into a dataframe named ‘df’.



```r
## created 10 vectors on 10 valu
temp1 <- rep(0, 10)     ## create
temp2 <- rep(0, 10)
temp3 <- rep(0, 10)
temp4 <- rep(0, 10)
temp5 <- rep(0, 10)
temp6 <- rep(0, 10)
temp7 <- rep(0, 10)
temp8 <- rep(0, 10)
temp9 <- rep(0, 10)
temp10 <- rep(0, 10)
## Create a dataframe of all ten
df <- data.frame(temp1, temp2, t
```