R Intermediate Homework 2

Caroline Tribble

The humble for loop is often considered distasteful by seasoned programmers because it is inefficient; however, the for loop is one of the most useful and generalizable programming structures in R. If you can learn how to construct and understand for loops then you can code almost any iterative task. Once your loop works you can always work to optimize your code and increase its efficiency.

Before attempting these exercises you should review the lesson R intermediate in which loops were covered.

Examine the following for loop, and then complete the exercises

```
data(iris)
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
```

```
sp_ids = unique(iris$Species)

output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(output) = sp_ids
colnames(output) = names(iris[ , -ncol(iris)])

for(i in seq_along(sp_ids)) {
    iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
    for(j in 1:(ncol(iris_sp))) {
        x = 0
        y = 0
        if (nrow(iris_sp) > 0) {
            for(k in 1:nrow(iris_sp)) {
                x = x + iris_sp[k, j]
                y = y + 1
            }
            output[i, j] = x / y
        }
    }
}
output
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Iris loops –

1. Describe the values stored in the object output. In other words what did the loops create?

The object output computes the mean sepal length, sepal width, petal length, and petal width for each species observed in the iris dataset.

2. Describe using pseudo-code how output was calculated, for example, Loop from 1 to length of species identities Take a subset of iris data Loop from 1 to number of columns of the iris data If … occurs then do …

Attempt at pseudo-code:

```
sp_ids = unique(iris$Species)
#sp_ids equals each unique type of species listed under 'Species' under the iris dataset.

output = matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(output) = sp_ids
colnames(output) = names(iris[ , -ncol(iris)])

#output equals a matrix with species ids as rows and columns from iris minus the last species column.
#Row names for output equal sp_ids and column names for output are the same as column names in the iris dataset.

for(i in seq_along(sp_ids)) {  #for i take the length from the length of sp_ids
   #subset iris where Species equals i in sp_ids and remove the extra Species column
   iris_sp = subset(iris, subset=Species == sp_ids[i], select=-Species)
    for(j in 1:(ncol(iris_sp))) { #j represents the column values in iris_sp
        x = 0
        y = 0
        if (nrow(iris_sp) > 0) { #if row value in iris_sp is greater than 0,then..
            for(k in 1:nrow(iris_sp)) {
                x = x + iris_sp[k, j]
                y = y + 1
            }
            output[i, j] = x / y
        }
    }
}
        #k represents the row values in iris_sp
        #iris_sp[row values, column values] represented here as [k, j] variables for dataframe
        #x is equal to x + iris_sp[k, j] where x represents the sum of samples
        #within that column for a specific species
        #y is equal to y + 1 where y represents the total number of samples within that column

output
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

```
        #output then computes a matrix of species_ids and the mean (x divided by y) for each
        #column (Sepal.Length, Sepal.Width, etc.) from the iris_sp dataset
```

3. The variables in the loop were named so as to be vague. How can the objects output, x, and y could be renamed such that it is clearer what is occurring in the loop.

The output could instead be named species.means, x could be sum.samples to represent the numerator in the mean, and y could be total1p to represent the denominator in the mean.

4. It is possible to accomplish the same task using fewer lines of code? Please suggest one other way to calculate output that decreases the number of loops by 1.

```
sp_ids <- unique(iris$Species)

species.means <- matrix(0, nrow=length(sp_ids), ncol=ncol(iris)-1)
rownames(species.means) = sp_ids
colnames(species.means) = names(iris[ , -ncol(iris)])


for(names in seq_along(sp_ids)) {
    iris_sp = subset(iris, subset=Species == sp_ids[names], select=-Species)
    for(n in names(iris_sp)) {
        if (class(iris_sp[,n]) == 'integer' | class(iris_sp[,n]) == 'numeric') {
            tot <- mean(iris_sp[,n])
            }
            species.means[names, n] <- tot
        }
    }
species.means
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa            5.006       3.428        1.462       0.246
## versicolor        5.936       2.770        4.260       1.326
## virginica         6.588       2.974        5.552       2.026
```

Sum of a sequence –

5. You have a vector x with the numbers 1:10. Write a for loop that will produce a vector y that contains the sum of x up to that index of x. So for example the elements of x are 1, 2, 3, and so on and the elements of y would be 1, 3, 6, and so on.

```
y <- NULL
x <- c(1,2,3,4,5,6,7,8,9,10)
  for (i in x) {
    y[i] = sum(x[1:i])
    }
y
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

6. Modify your for loop so that if the sum is greater than 10 the value of y is set to NA

```
y <- NULL
x <- c(1,2,3,4,5,6,7,8,9,10)
for (i in x) {
  y[i] = sum(x[1:i])
  if (y[i]>10) {
    y[i] <- NA
  }
}
y
```

```
##  [1]  1  3  6 10 NA NA NA NA NA NA
```

7. Place your for loop into a function that accepts as its argument any vector of arbitrary length and it will return y.

```
sumofseq = function(n) {
y <- NULL
for (i in n) {
  y[i] = sum(n[1:i])
  }
  y
}

x <- c(1,2,3,4,5,6,7,8,9,10)
z <- c(1,2,3,4,5)
w <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)

sumofseq(x)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
sumofseq(z)
```

```
## [1]  1  3  6 10 15
```

```
sumofseq(w)
```

```
##  [1]   1   3   6  10  15  21  28  36  45  55  66  78  91 105 120
```

Fibonacci numbers and Golden ratio –

Fibonacci numbers are a sequence in which a given number is the sum of the precedding two numbers. So starting at 0 and 1 the sequence would be 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Write and apply a simple R function that can accomplish this task with a for loop. Then write a function that computes the ratio of each sequential pair of Fibonacci numbers. Do they asympotitcally approch the golden ratio (1 + sqrt(5)) / 2) ?

Fibonacci function:

```r
fib_funct <- function(n) {
    if (n==1) return(0)
    x <- c(0,1)
    while (length(x) < n) {
        i <- length(x)
        j <- x[i] + x[i-1]
        x <- c(x,j)
    }
    return(x)
}
fib_funct(13)
```

```
##  [1]   0   1   1   2   3   5   8  13  21  34  55  89 144
```

Golden Ratio Function:

```r
gold_funct <- function(n) {
    if (n==1)
    x <- c(0,1)
    while (length(x) < n) {
        i <- length(x)
        j <- fib_funct(i)/fib_funct(i+1)
        x <- c(i:j)
    }
    return(x)
}
```

I tried the golden ratio function but to no avail.