R intermediate

Dan McGlinn

January 15, 2016

> Premature optimization is the root of all evil – Donald Knuth

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)) {
                trait_sum = x + iris_sp[k, j]
                row_count = y + 1
            }
            output[i, j] = x / y
        }
    }
}
output
```

```
##            Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa              NaN         NaN          NaN         NaN
## versicolor          NaN         NaN          NaN         NaN
## virginica           NaN         NaN          NaN         NaN
```

Excercises

Iris loops

1. Describe the values stored in the object `output`. In other words what did the loops create? The loops created a mean value for the iris'. i/x is the sum of the iris and j/y is the count of the iris. When divided, this gives you the mean.

2. Describe using pseudo-code how `output` was calculated, for example,

```
loop through ...
   loop through ...
      And so on ...
```

when determining the output, which is ultimately the mean of the iris', you are creating a ration of the sum and count of the iris. This can be done by setting i equal to seeq_along(sp_ids) and j equal to 1:(ncol(iris_sp)). We then have that if nrow(iris_sp) is greater than 0 for k is equal to 1:nrow(iris_sp). This is where we arrive at the output formula of [i,j] where it is equal to the ration x/y. 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. output could be renamed as iris_mean, x could be iris_sum, and y could be iris_count

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. You can reduce the number of loops by 1 by removing "y"/row count since that value is already determined by nrow(iris_sp) which gives us 50 both times.

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 = 1
for (i in 2:10){
  y[i] = y[i - 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 = 1
for (i in 2:10){
  if (!is.na(y[i - 1])) {
     y[i] = y[i - 1] + i
     if (y[i] > 10)
         y[i] = NA
  }
}

y
```

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

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

```
cum_sum_cutoff = function (x, cutoff = 10){
    if (!is.vector(x))
        stop ('x must be a vector')
    if (!is.numeric (x))
        stop ('x must be numeric')
    y = NULL
    for (i in 1:length(x)){
        y[i] = sum(x[1:i])
        if (y[i] > cutoff){
            y[i] = NA
        }
    }
    return(y)
}
cum_sum_cutoff(1:10)
```

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