Thinking Like a Computer: Pseudo-coding & Functions

ACTIVITY 18

Sometimes it is helpful prior to tackling writing code to spend just a minute and think about the problem and then jot down the general steps that you need to accomplish in order to complete the task. This pseudo-code need not be written in syntax but can provide a map that helps you to accomplish the code. The key to successfully turning your map into code will be to link these steps with what you know about R so far. The other key is to recognize that a computer script will only do what you tell it! We will also write our own functions in this activity.

Purpose

The purpose of this activity is to help facilitate thinking like a computer (i.e., in logical step sequence). You will also reinforce the looping logic and vector applications by writing the code for yourself.

Learning Objective

- Step-by-step thinking and analysis
- Pseudo-coding
- Writing functions

Required Resources

- R, R Studio
- TriMet dataframe

Time Allocated

50 minutes in class

Tasks

This activity begins assuming R Studio is open.
If not, please start R Studio and open the sample script for the activity (if provided).

A. Control Structures

Control structures such as loops and if-then statements are easy ways to control the execution of a set of commands. R's indexing of vectors make looping through code to generate plots very easy. Dalgaard has a quick summary on page 336.

Loops are the first syntax:

```
-----------------------------------------------------
# 5 - Control Structure
#-----------------------------------------------------
# Loops
#++++++++++++++++++++++++++++++
for ( j in 1:4){
print (j)
}
```

This loop will execute the code between the `{}` four times. Each time `j` gets incremented when step (e.g., `j=1, j=2, j=3, and j=4`).

Note you do not need to initialize the loops with a `j=1` or add a `j+1` at the last step nor does the code after the `in` statement need to be sequential. Let's look at this `for` statement:

```
x <- 1:10
```

```
y <- 1:10
col.list <- c("red", "blue", "green")
par (mfrow=c(1,3))

for ( k in col.list ) {
       plot (x,y, col=k, main=k, pch=16)
     }
```

The table shows the number of steps in the loop and the value of `k` in each step. See how the value of `k` changes based on the loop step? The resulting plot is shown below (Figure 35).

	Start of Loop	Step 1	Step 2	Step 3
Value of k	null	red	blue	green
col.list		col.list[1]	col.list[2]	col.list[3]

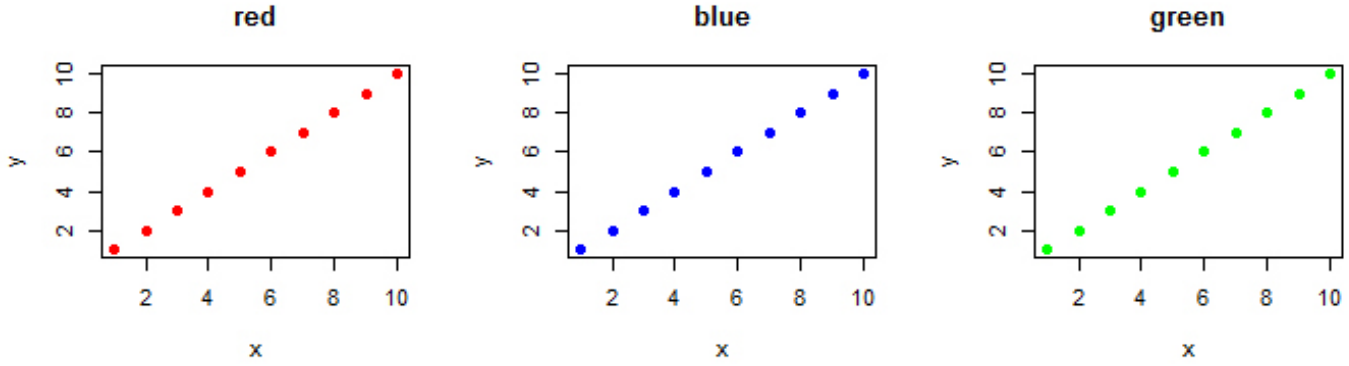

Figure 35 Plot outputs from sample loop code

Conditional statements such as `if-then` are the next most common control structure. R's syntax for defining your own function in code looks like this;

```
#if then conditions
#++++++++++++++++++++++++++++++
#if (condition is TRUE){
# do
# } else {
#do this
# }

mileposts<- c(22.1, 25.2, 43, 29.1,29.2,26.3)

for (j in 1:length(mileposts)) {
    if (mileposts[j]<27) {
      print("too slow")
    }   else {
      print("too fast")
```

```
    }
  }
```

In this code, R loops three times and evaluates `mileposts[j]`. If it is less than 27, it prints the logical evaluation of `mileposts[j]`. If this condition is not met, it executes the code in the `else {}` braces. A common error is not including the correct matching braces. R Studio highlights the matching pairs.

B. Functions

You have already seen built-in functions in R such as:

```
plot(x,option1=1, option2=2,option3=3)
```

The plot function takes the input `x` and returns a plot. Any options are also specified and passed to the function.

It is easy to write your own function in R. R's syntax for defining your own function in code looks like this:

```
testfunc <- function(x,y){
                    z <- x + y
                    return (z)
}
#testfunc is created (see object appear in R studio objects)
testfunc( 3600,86400 )
```

Here *testfunc* is created as a function. It takes inputs `x` and `y`, executes the code enclosed by `{ }` and returns the output of `z`. The return is optional, but is required if you have something specific you want the function to return. This is because the value of `z` is not stored by R (it will not show in your R Studio workspace, it is internal to the function operation).

A function will be helpful if you are planning to repeat a set of code again and again. It is helpful to first write working code, identify the variables you might want the function to take as input, then create the function.

C. Good Practice: Write Psuedo Code

I often find it is helpful to spend just a minute and think about the code I am trying to write. If you remember to think like a computer this will easily translate to code. Obviously the program will only do what you tell it.

Let's return to the code you wrote for the last activity using the data frame *trimet*.

1. Create a new R script file for this activity, copy the commands to load the *trimet* data frame and load the data from the course website.

You've been asked to create a function that creates plots for an entire day that show (1) pattern distance vs. load, (2) pattern distance vs. dwell, (3) pattern distance vs. max speed. You need to create these 3 plots for AM peak (6AM-9AM), off-peak (9AM-4PM), and PM peak (4PM-7PM). The user of your function will also want to specify direction.

If I wanted to write the pseudo-code it might look something like this :

1. Read in the *trimet* data table
2. Subset data frame
    a. To user specified date (*service_day*) and to user specified (*direction*).
        i. To AM Peak (*leave_time >= 6*3600 and < 9*3600*)
        ii. To Mid Day (*leave_time >= 9*3600 and <= 16*3600*)
        iii. To PM peak (*leave_time >= 16*3600 and <= 19*3600*)
3. Arrange a 3 column by 3 column plot window
4. *Check the y and x ranges of each so the plot will be on the same scale.*
5. Plot
    a. From data set (i), pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for AM peak.
    b. From data set (ii), Plot pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for mid-day
    c. From data set (iii), pattern_dist vs est_load, pattern_dist vs dwell, pattern_dist vs max_speed for PM peak.
6. Label the plots with *service_day* and *direction*

When I first translate my code and look at the plots, I notice that I missed step 4. If I don't check the range for entire data set, I won't be able to easily compare the plots for each day. I have a decision to make: Do I want the ranges to vary by day or should i hard code them so that each day looks the same? Right now, I opt for each day to be different.

The base code looks like this:

```
day <- subset(trimet, service_day==”2007-03-08” & direction==0)
am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

#Find the range for the plots, hard code some after inspection
x.dist <- range (day$pattern_dist)
y.load <- range (day$est_load)
y.dwell <- c(0,200)
y.speed <- c(0,60)

par (mfrow=c(3,3), oma=c(1,1,3,1)) #set up 3x3 plot window and create
an outer margin to label all the plots with oma
plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
plot (am$pattern_dist, am$dwell, main=”AM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
plot (mid$pattern_dist, mid$dwell, main=”Mid-day”, xlim=x.dist, ylim=y.
dwell)
plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
plot (pm$pattern_dist, pm$dwell, main=”PM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

#label the plot sets
mtext (line=0, “service_day==2007-03-08 & direction==0” , outer=TRUE,
font=2)
```

You could make a few more tweaks but seems to be useful and working for one day and direction.

To turn this into a function, what are the things that will need to change for each set of plots?

Circle those items in the code above before going on to the page.

What needs to be changed is highlighted:

```
day <- subset(trimet, service_day==”2007-03-08” & direction==0)
am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

#Find the range for the plots, hard code some after inspection
x.dist <- range (day$pattern_dist)
y.load <- range (day$est_load)
y.dwell <- c(0,200)
y.speed <- c(0,60)

par (mfrow=c(3,3), oma=c(1,1,3,1)) #set up 3x3 plot window and create
an outer margin to label all the plots with oma
plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
plot (am$pattern_dist, am$dwell, main=”AM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
plot (mid$pattern_dist, mid$dwell, main=”Mid-day”, xlim=x.dist, ylim=y.
dwell)
plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
plot (pm$pattern_dist, pm$dwell, main=”PM Peak”, xlim=x.dist, ylim=y.
dwell)
plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

#label the plot sets
mtext (line=0, “service_day==2007-03-08 & direction==0” , outer=TRUE,
font=2)
```

So, we just have to figure out how to get R to the subset by accepting a variable as the logic for the subset. Fortunately, this is easy:

```
day.plt <- ‘2007-03-08’
dir.plt <- 0
test <- subset(trimet, service_day==day.plt & direction==dir.plt)
```

Now let's rewrite our code as a FUNCTION

```
trimet.plot <- function (df, day.plt, dir.plt) {
  day <- subset(df, service_day==day.plt & direction==dir.plt)
  am <- subset(day, leave_time >= 6*3600 & leave_time< 9*3600)
  mid <- subset(day, leave_time >= 9*3600 & leave_time < 16*3600)
  pm <- subset(day, leave_time >= 16*3600 & leave_time < 19*3600)

  x.dist <- range (day$pattern_dist)
  y.load <- range (day$est_load)
  y.dwell <- c(0,200)
  y.speed <- c(0,60)

  par (mfrow=c(3,3), oma=c(1,1,3,1))
  plot (am$pattern_dist, am$est_load, xlim=x.dist, ylim=y.load)
  plot (am$pattern_dist, am$dwell, main="AM Peak", xlim=x.dist, ylim=y.
dwell)
  plot (am$pattern_dist, am$max_speed, xlim=x.dist, ylim=y.speed)

  plot (mid$pattern_dist, mid$est_load, xlim=x.dist, ylim=y.load)
    plot (mid$pattern_dist, mid$dwell, main="Mid-day", xlim=x.dist,
ylim=y.dwell)
  plot (mid$pattern_dist, mid$max_speed, xlim=x.dist, ylim=y.speed)

  plot (pm$pattern_dist, pm$est_load, xlim=x.dist, ylim=y.load)
  plot (pm$pattern_dist, pm$dwell, main="PM Peak", xlim=x.dist, ylim=y.
dwell)
  plot (pm$pattern_dist, pm$max_speed, xlim=x.dist, ylim=y.speed)

  #label the plot sets
   mtext (line=0, paste("service_day==", day.plt, " & direction==", dir.
plt) , outer=TRUE, font=2)
}
```

This function takes three inputs (**df**, **day.plt**, **dir.plt**) and assigns them to these respective values in the function code. You can follow this in the code with color coding. Now to create our six plots, all that is required is:

```
trimet.plot (df=trimet, day.plt='2007-03-05', dir.plt=0)
```

Experiment with changing the date and direction. We will work as a class to turn this into a loop!

Deliverable

None, but save your R script file.

Assessment

None