Session 8: Function Writing

Samuel P Callisto

August 2, 2018

```
## load packages
library(testthat)
library(tidyverse)
```

Function Writing

A trivial example

```
sumMinusOne <- function(x){
  output <- 0
  for(i in 1:length(x)){
    output <- output + x[i]
  }
  return(output-1)
}

sumMinusOne(2:4)
```

```
## [1] 8
```

When would I ever use this?

Example 1: Wrapper function

Problem: importing files with multiple headers causes the data type to be interpretted incorrectly, causing resulting in manual typecasting for multiple rows (annoying!)

```
## Excel files
excel <- read.csv("datasets/TPM_sim_dataset_20180607.csv", as.is = T, stringsAsFactors = T, header = T)

## dataset imported using read.csv()
str(excel)
```

```
## 'data.frame':    30 obs. of  4 variables:
##  $ subjectid: chr  "SID" "3" "4" "6" ...
##  $ var1     : chr  "m/d/y" "5/4/2018" "5/5/2018" "5/6/2018" ...
##  $ var2     : chr  "hh:mm" "5:40" "5:41" "5:42" ...
##  $ var3     : chr  "ug/mL" "0.156" "0.157" "0.158" ...
```

```
excel$subjectid <- as.integer(excel$subjectid)
excel$var3 <- as.double(excel$var3)
```

Solution: write a wrapper function that assigns correct header row while maintaining data types.

```
headr <- function(file, header.row=1, data.start=3, ...){
  headers <- read.csv(file = file, skip=header.row-1, header = F, nrows = 1, as.is = T)
  dataset <- read.csv(file=file, skip = data.start-1, header = F, as.is=T, ...)
```

```
  names(dataset) <- headers
  return(dataset)
}

## import same file using headr wrapper function
topiramateData <- headr("datasets/TPM_sim_dataset_20180607.csv", stringsAsFactors= T)

## dataset imported using headr()
str(topiramateData)
```

```
## 'data.frame':    29 obs. of  4 variables:
##  $ subjectid: int  3 4 6 7 11 16 35 38 39 40 ...
##  $ var1     : chr  "5/4/2018" "5/5/2018" "5/6/2018" "5/7/2018" ...
##  $ var2     : chr  "5:40" "5:41" "5:42" "5:43" ...
##  $ var3     : num  0.156 0.157 0.158 0.159 0.16 0.161 0.162 0.163 0.164 0.165 ...
```

Notice how few arguments need to be filled out manually each time you import a file using the helper function since you are allowed to set your defaults.

Example 2: Dealing with Times

Problem: RedCap stores my times as a character string ("6:45"), but I want to calculate the difference between observations.

Solution: write a function to use this time and the next time you encounter clock times

```
numericTime <- function(vec){
  ## separate hours and minutes into a vector
  sapply(strsplit(vec,":"),
         function(x) {
           ## convert to numeric type to allow arithmetic operations
           x <- as.numeric(x)
           ## numeric time = hours + (minutes/60) rounded to two decimals
           round(x[1]+x[2]/60,2)
         }
  )
}

topiramateData$TIME <- numericTime(topiramateData$var2)
```

A general rule of thumb:

If you find yourself copying code within or between files, you should probably just write a function. Better still, add functions to a personal R package that you can easily import and share.

Great resource for learning more about writing functions

http://adv-r.had.co.nz/Functional-programming.html

Unit testing

How can you check if your function is doing what you think it is? Let's go back to our sumMinusOne() function

A trivial example revisited

```
test_that("single value minus one",{
  expect_equal(1, sumMinusOne(2))
})

test_that("vector sum minus one",{
  expect_equal(5, sumMinusOne(1:3))
})

test_that("characters shouldn't work",{
  expect_error(sumMinusOne("test"))
})
```

Testing our wrapper function for read.csv()

```
test_that("ID imported as integer",{
  imported <- headr("datasets/TPM_sim_dataset_20180607.csv")
  expect_true(is.integer(imported[1,1]))
})

test_that("time imported as character",{
  imported <- headr("datasets/TPM_sim_dataset_20180607.csv")
  expect_true(is.character(imported[1,3]))
})

test_that("concentrations imported as numeric",{
  imported <- headr("datasets/TPM_sim_dataset_20180607.csv")
  expect_true(is.numeric(imported[1,4]))
})
```

Testing numericTime()

```
test_that("numbers shouldn't work",{
  expect_error(numericTime(5.37))
})

test_that("strings without colon shouldn't work",{
  expect_true(is.na(numericTime("546")))
  expect_true(is.na(numericTime("5.46")))
})

test_that("single digit & double digit times should work",{
  expect_equal(5.5, numericTime("5:30"))
  expect_equal(12.0, numericTime("12:00"))
```

```
  ## notice we don't test for real clock times
  ## we could add this into the function later
  expect_equal(25.75, numericTime("25:45"))
})

test_that("differences in time can now be calculated", {
  expect_equal(2, numericTime("5:00") - numericTime("3:00"))
  expect_equal(4.5, numericTime("4:30"), numericTime("12:00"))
})

test_that("vectorized application succeeds",{
  expect_equal(c(12, 1.5), numericTime(c("12:00", "1:30")))
})
```