Code Review

Jared Bennett

22 September, 2019

Homework 1 Notes and Code Review

This section is mostly a focus on reviewing other students solutions, using PS1 as an example. We'll begin with things that I saw in homeworks, anything that didn't make sense to people, and then finish section with the code review.

PS1 Notes

1. **SUBMISSIONS MUST BE A PDF DOCUMENT**
   Specifically, you must knit the Rmd, so that your code builds, your document builds, and all of the output is shown. Saving the .Rmd as a pdf is not the same.
2. `suppressWarnings()`
   Don't use this. Fix the problems.
3. Read the function documentation
   R has very complete function documentation, in terms of what options exist and what they do. This is easily accessed by using `?write.table()` or `??write.table()`. This also provides the default values for all of the options.
4. Scoping
   We will cover this in detail in unit 5, but here's a short explanation.
   R looks for variables within the environment where they are called, ie. within a function. If it doesn't find it, it then looks in the parent environment. eg.

```
# variable defined in .Global
x <- 1:10

myFunc <- function(){
  # variable defined in myFunc
  y <- 1:10

  # return
  return(x + y)
}

myFunc()
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

   This occurs again when using `<-` vs `<<-`.
   (Potentially interesting SO discussion.)

```
# variable defined in .Global
x <- 1
y <- 1
# z <- 1

# function
```

```r
sideEffects <- function(){
  # local assignment
  y <- 2

  # global assignment
  x <<-2

  # global creation
  z <<- 10
}

# run function
sideEffects()

# see results
x; y; z;
```

```
## [1] 2
```

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

```
## [1] 10
```

5. `suppressPackageStartupMessages()`
   Do use this. This function (should) suppress messages from loading a package. This prevents the long messages from things like `tidyverse`. **Beware** - This may cover up warnings about function name clashes.

6. `require()` vs. `library()`
   See the bCourses anouncement Additionally, I would never put them in functions. I prefer them near the top of the document (that way you know what you need immediately), or at least at the top of the chunk that requires them. Within functions, it is best to call things using the namespace explicitely, ie. `devtools::build()`. The prevents you from loading all of the functions in a package and cluttering the namespace.

7. Beware base function names. Don't redefine names. ie, `data` or `list`.

8. Use `return()` explicitly in a function

   ```r
   myFunc <- function(x){
     # crazy stuff

     # explicit return
     return(x)
   }
   ```

   Also, `print()` is not the same as `return()`. You can't use the output of print (easily), while a returned object can be stored for later use.

9. if/else Statements
   I don't know why this works in functions, but it's not correct.

   ```r
   # basic if statement
   if(TRUE){
     cat("I did the thing!\n")
   }
   ```

   ```
   ## I did the thing!
   ```

```r
# properly formatted if/else statement
if(FALSE){
  cat("I did the thing!\n")
} else {
  cat("I like cats\n")
}
```

```
## I like cats
```

```r
# improperly formatted if/else statement
if(FALSE){
  cat("I did the thing!\n")
}
else {
  cat("I like cats\n")
}
```

```
## Error: <text>:5:1: unexpected 'else'
## 4: }
## 5: else
##    ^
```

```r
# improper if/else inside function
printFunc <- function(){
  if(FALSE){
    cat("I did the thing!\n")
  }

  else {
    cat("I like cats\n")
  }
}

# function works
printFunc()
```

```
## I like cats
```

Code Review

Today we will practice paired code review for PS1. In order for this of benefit, you will need to be as honest with your partner as possible. If something is actually hard to follow, tell them! If you thought that something they did was clever, also tell them!

Procedure

You will each spend 10 minutes reading your partner's code while he/she will be available to answer questions you have about their work. I will float around and can make any clarifying points. After that, we will go through Chris' solution to problem 4b and see what makes sense or what could be made clearer.

Some guiding questions

1. Is the code visually easy to break apart? Can you see the entire body of the functions without scrolling up/down left/right? The general rule-of-thumb is no more than 80 in either direction (80 character width, 80 lines. The first restriction should be practiced more religiously)
2. Are there "magic numbers" present in the code? i.e. Hard-coded values or indices. In this assignment, some indices are acceptable given the structure of the scraped page, but they should be accompanied by some documentation about the assumed structure of the data.
3. Is it clear why each variable is being created? Names are good guidance here.
4. Is the code separated into logical "steps"? Or is it just one big blob? If you have descriptive function names, then you can get away without comments. Otherwise, each chunk of code should have a short comment explaining its purpose.
5. Are the input tests reasonable? Do they seem to catch every case that came to your mind as you wrote your function?