Brainstorming an App for Teaching Coding

Jay Emerson

January 2022

The Premise

Code that produces the right answer isn't enough. Most of what I've seen in S&DS and CPSC focuses on the theory, methodology, and (perhaps) exercises and questions that are objectively evaluated for only the correctness of a solution rather than the quality of a solution. I think that quality of code, efficiency (of effort and of code), and best practice for collaborative code/research are all equally important. A starting point is at the introductory level, but this extends well beyond that in our curriculum.

The Realization (updated February 16, 2022 by Jay)

Objective grading should be a do-able first dimension. Judging the "similarity" between a new submission and a library of candidate solutions also seems do-able based on early explorations with package `similaR`. Finally, code style checks provide a third dimension for evaluation, with packages `lintr` and `styler` on our radar. But other style points are likely not supported by these packages and will require some coding. Beyond these three dimensions, the task of deploying efficiently at scale is a substantial software development problem.

The Language

My focus is R, but the idea isn't language-specific. I should probably try to pilot this project in parallel with R and Python.

An example (in R)

```r
NR <- 4
NC <- 100000
x <- matrix(sample(1:10, NR*NC, replace = TRUE), NR, NC)

# Challenge: calculate the column means; save the answer
# in a vector of length 'NC' called 'ans'.  Below, I'll use
# 'ans1', 'ans2', ... just for the purpose of this example.
```

Candidate answers below

```r
# Solution 1: Very computationally inefficient; copies made as ans1 grows.
ans1 <- c()
for (j in 1:NC) {
  ans1[j] <- mean(x[,j])
}

# Solution 2: Much better than Solution 1; ans2 is simply "filled up".
ans2 <- rep(0, NC)
for (j in 1:NC) {
  ans2[j] <- mean(x[,j])
}

# Solution 3: Essentially equivalent to Solution 2 and should be
#             easily recognized by intermediate R coders.  Some people
#             mistakenly think this is faster than Solution 2, however.
ans3 <- apply(x, 2, mean)

# Solution 4: Could be much faster in some (larger) examples, but this
#             doesn't extend beyond a small set of basic functions.
ans4 <- colMeans(x)

# Solution 5: Same as Solution 2, but without proper indentation.
ans5 <- rep(0, NC)
for (j in 1:NC) {
ans5[j] <- mean(x[,j])
}

# Solution 6: About as confusing, inefficient, and unreadable as possible.
ans6 <- c()
for (j in 1:NC) {
temp <- 0
for (i in 1:NR)
temp <- temp + x[i,j]
temp <- temp / NR
ans6[j] <- temp
}
```

R packages/functions to explore

- https://rdrr.io/cran/SimilaR/
- foreach. See, for example, https://github.com/RevolutionAnalytics/foreach/blob/master/R/getsyms.R
- codetools. https://homepage.cs.uiowa.edu/~luke/talks/DSC07.pdf
- knitr of course, https://deanattali.com/2015/03/24/knitrs-best-hidden-gem-spin/
- gradeR. https://cran.r-project.org/web/packages/gradeR/vignettes/gradeR.html
- exams. Bettina and Achim, probably a different area
- knitr. https://bookdown.org/yihui/rmarkdown-cookbook/purl.html

References

I found quite a bit about qualify of large-scale coding projects, but that isn't really my focus.

- https://link.springer.com/chapter/10.1007%2F978-3-540-73460-4_26
- https://codescene.com/blog/evaluate-code-quality-at-scale/
- https://www.diva-portal.org/smash/get/diva2:1461671/FULLTEXT01.pdf

This seems closer to the mark, but they will try to sell something:

- https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality
- https://dzone.com/articles/7-best-python-code-review-tools-recommended-by-dev

I should look at these in greater detail:

- https://www.stgm.nl/quality/stegeman-quality-2016.pdf
- https://realpython.com/python-code-quality/

Nice – the following acknowledges that the hard part is the most subjective and "requires manual inspection of code from an experienced programmer" – which is what I'm hoping to address in order to scale this to a large class.

- https://arxiv.org/pdf/1509.03556.pdf

A question/answer from StackExchange that I liked

Question: How can I determine if the code sample they've provided me is efficient and of good quality?

Answer: The things you will not be able to evaluate are correct use of language idioms and library usage. So, these are not things you should try to look at.

What you can evaluate is:

- How well structured the code looks like
- Well named variables (can you make sense of things)
- Well composed functions/units of code
- Consistency in the code base

The above points (though not exhaustive) will indicate whether the code smells or not and should be something an experienced programmer can identify as being good or bad.

In short - look for things that should indicate good code regardless of language.

From Jay: I would add something about documentation if it is more than a trivial, obvious example, but perhaps that could be included in "well structured".

A further comment I found: Another important one: "Are the comments clear, meaningful, and easy to understand?" Could you learn a little bit about what a piece of code does from the comments, even if you have little exposure to the language?

Another perspective: Comments? What are those? Seriously though, code should be self commenting. Comments should only be there to explain the why or give reasons for bad code.

Homework for February 4, 2022

Assemble a list of measures or characteristics of code that we think ought to be useful or relevant for possibly measuring/learning about code similarity. Examples:

- Number of explicit for loops
- Use of nested loops
- Number of temporary/incidental variables created
- Total size of temporary/incidental variables created
- Number of lines of code not including blank lines
- Number of conditional statements (perhaps if separate from else)
- Number of function calls (and we're open to actually counting specific function usage)
- Use of functions in a list of "likely" functions for this problem
- Number of assignments (different from number of lines of code)
- Number of conditional statements, including `ifelse()`
- Benchmarking (speed/memory)?
- Creation of a function (explicit)
- Creation of a function (implicit)
-

Do a "literature search" of the R Project Universe for useful things so we don't reinvent the wheel. Example: a code parser that would make it easier to do work on all function calls present, and all symbols/objects created, etc…

codetools, foreach, parse()

https://cragkhit.github.io/publications/chaiyong_icsme2016ds.pdf

Real "literature search" of evaluating code similarity. Software plagiarism?

https://educationaldatamining.org/EDM2021/virtual/static/pdf/EDM21_paper_96.pdf

https://rdrr.io/cran/SimilaR/

https://journal.r-project.org/archive/2020/RJ-2020-017/RJ-2020-017.pdf