Assessing data analysis and programming

Hadley Wickham, Garrett Grolemund

September 15, 2011

Abstract

Modern data analysis is impossible to do with programming, and so we must teach programming as part of the statistics curriculum. Both data analysis and programming demand different approaches to teaching than the usual mathematical statistics course. This paper outlines techniques that we have found successful, focussing particularly on the challenging issue of assessing student performance.

1 Introduction

Data analysis, the craft of turning data into knowledge, insight and understanding, is a critical skill for statistics graduates. It weaves together the mathematical and computational threads that span the curriculum and provides practical tools for working with data. Modern data analysis demands the use of a computer: pencil and paper are no longer adequate for the quantity of data we must deal with. Despite these facts, classes that teach both data analysis and programming are rare (Nolan and Temple Lang, 2010). The aim of this paper is to help make them more common by focussing on a challenge faced when teaching data analysis and programming: how do you effectively assess student performance?

This paper has been shaped by our experience teaching Stat405, a data analysis class at Rice University, `http://had.co.nz/stat480`). Stat405 grew out of an earlier course taught at Iowa State, and combined, we have taught it eight times in the last six years. The primary focus of the class is data analysis, but because data analysis is impossible without the right tools, a secondary focus is programming. Programming is often a dirty word in statistics, but mastery of the basics is just as important as mastery of the mathematical underpinnings of statistical theory. That said, while the course uses computational tools extensively, it is not computational: it focusses on using the tools to do data analysis, not on learning about computation. A brief overview of the topics currently covered in class is provided in Appendix A.

While there's no argument that modern data analysis must be performed on a computer, there is a choice between learning a graphical user interface (GUI) or a programming language. We believe that learning how to program is a vital skill for every analyst who will be working intensely with data. While convenient, a GUI is ultimately limiting and hampers three properties essential for good data analysis: **reproducibility**, the ability to exactly recreate a past analysis, which is crucial for good science; **automation**, the ability to rapidly re-create an analysis when data changes; and **communication**: code is just text, so it is easy to put in an email and ask for help, or print out and have graded.

Compared to math-centric classes, data analysis and programming require rather different means of assessment. Data analysis is a high level skill, and there is typically no right answer (although there are better and worse answers). Section 2 discusses these challenges and outlines our current approach, which includes a transition from small individual analyses to large team projects, and grading on dispositions, not just knowledge or skills. Grading programming seems deceptively easy: you just check that the results are correct. But incorrect results give the student no insight into what needs to be improved, so it is better to also assess the process. Section 3, we will argue that code is a medium of communication and should be graded as such. Our approach to grading code draws more from composition than from statistics, grading a program like you might grade a paper.

2 Data analysis

Data analysis is a high-order, creative skill. It requires the mastery of tried and true techniques as well as the ability to create new variations to address the problem at hand. Data analysis is a craft, a combination of science and art, and can not be taught with the same techniques we use for more mathematical topics. One particular challenge with teaching and assessing data analysis is the lack of a formal model. While many have proposed models based on personal experience (Cox, 2007; Tukey and Wilk, 1966; Huber, 2011; Chatfield, 1995; Box, 1976; Wild and Pfannkuch, 1999), there has been little work developing and validating a comprehensive model. Others have claimed that data analysis can have no systematic exposition, and can only be taught through apprenticeship (Huber, 2011). While apprenticeship is generally a excellent way to learn any field, it does not help the problem we face: scaling up to meet the increasing demand for skilled analysts.

The missing framework for data analysis makes teaching challenging because we have no infrastructure to guide students and to suggest metrics for grading. In the absence of a solid foundation, we fall back on an apprenticeship-inspired approach, giving the students many opportunities to practice data analysis and receive detailed feedback on their work. This is a lot of work, but doable for a small class.

Overall assessment is divided into weekly homeworks and three larger team projects. Each project and the first four homeworks focus on data analysis. The homeworks provide scaffolding for students apply their new skills to a small, relatively well defined problem, and to receive rapid feedback. The homeworks are open-ended (find x interesting plots of dataset y), and the grading focus is on providing copious feedback. This helps students learn our expectations for a good data analysis. Early homeworks can be frustrating for students because they have so many questions to ask, but don't yet have the skills to answer them. This helps motivate the rest of the course: students are gradually empowered to answer the more sophisticated questions they've been thinking about the whole time.

To challenge students in an environment similar to the working conditions of practicing statisticians, the three team projects allow students to tackle larger problems. These are still open-ended but require considerably more work, as students need to write a 10-15 page report and integrate work across team members to provide a consistent narrative arc. The team projects have been successful mainly because we have followed the excellent guidelines of Oakley et al. (2004): assign teams to ensure diversity, teach tools for dealing with team conflict, and adjust individual grades based on team citizenship.

The first project uses data already seen in class and in homeworks, and class time is used to discuss team work and provide feedback on early drafts. The second project increases the challenge by providing a new dataset and no in-class scaffolding. Students need to independently generate interesting questions and answer them on their own. The final project increases the challenge still further by requiring students to select their own data set. This also gives students an opportunity to practice their data cleaning skills and gain some experience with the challenges of finding good data.

All three projects result in written reports (produced in latex) that describe the data, analysis and results. The final project is also presented at a poster session. Class time is used to teach both report writing and poster presentation since communication is such an important skill for apprentice statisticians to learn. The poster session has also been successful at raising the profile of the class within the university, and it's a fun opportunity for the students to dress up and talk about their work.

As data analysis assessment shifts from homeworks to projects and the amount of project scaffolding decreases, the focus of lecture materials and weekly homeworks also shifts. The first few weeks of class are tightly choreographed to give students the absolute essentials of visualisation and manipulation in the least amount of time. Many in-class examples show how to string new techniques together to uncover interesting facts about the data. Later classes focus more on abstract technical skills such as function writing, and the students are left to figure out how to best apply them. This approach develops student autonomy as the semester unfolds. Students mature from imitating what they see in lecture at the beginning of the semester to guiding themselves through every step of the data analysis process by the end of the semester. Accordingly, later homeworks focus on technical skills, helping students to master the programming techniques they need to tackle more interesting questions in the projects. This shift also tends to make later homeworks less time consuming, because they are more concrete, which balances the increased time commitments of the projects.

Data analysis homeworks and projects are graded using a rubric of three components: curiosity, scepticism and organisation. These reflect three key dispositions of a statistician (Wild and Pfannkuch, 1999): they should be **curious** about data and able to creatively apply old tools in new ways; they should be **sceptical** about their findings, always aware that a result may be the result of chance alone and always on the look out for a way to double check their work; and they should be able to present their findings in an **organised** manner that guides the audience from raw data to results. A copy of the complete rubric is available in Appendix C, as well as a selection of anonymous graded homeworks (published with student permission).

An interesting side affect of this rubric is that because the same rubric is used throughout the semester, early assignments tend to receive very low grades (many 2s and 3s out of 5). Copious reassurance is given in class to ensure that students realise that this won't negatively effect their final grade, but it does seem to provide considerable incentive for Rice students to invest time in the homework.

3 Programming

When assessing programming it is easy to focus on what's easy to assess: results. With some investment it's possible to automate grading, even automatically providing helpful comments if the answer is close (Murrell, 2008). There are two problems with this approach: if the code is incorrect the student gains no insight into possible causes, and it encourages a mindset focussed on what works right now, instead of what works now and in the future.

The approach that we use, and recommend, is to centre assessment around the idea of computer code as an artefact of communication. Code communicates not only to the computer, but also to collaborators and even the original author six months after they have written the code. Clarity of communication is so important because good code does more than just return the correct answers. Good code continues to provide the right answers even as requirements change; clearly conveyed intent makes it much easier to adapt code to future needs.

We assess code on three criteria: planning, execution and clarity. **Planning** grades evidence of thought before writing the code. Is there a clear strategy, described by an introductory comment? Does the breakdown of the large problem into smaller pieces make the solution simpler? **Execution** grades mastery of R vocabulary and use of functions: ideally the code should be concise and free of duplication. **Clarity** grades how easy it is to read and understand the code. Do function names suggest their purpose? Are comments well-integrated and do they explain the why, not the how? One reference that we have found helpful for developing the rubric is Kernighan and Plauger (1978): while some advice is dated, the majority is timeless advice on how to write clear, elegant code.

Coupled with these high level objectives are penalties for poor style. Students need to learn the stylistic conventions for writing code, just as they learn punctuation conventions for writing prose. Style is someways simultaneously both trivial and crucial: it is unrelated to the quality of the underlying ideas, butproperuseofconventionsmakesideasmucheasiertounderstand! Points are deducted for mistakes like incorrect spacing/indenting, overly long lines or confusingly named functions or variables. This makes the code much easier to read (and thus grade) and helps to establish a common style among students, which also facilitates collaboration. A copy of the complete rubric is included in Appendix C.

A side benefit of the struggle to teach good programming is that our own code has become significantly easier to understand. This inspired me to try another technique which has proved useful: provide students with (anonymised) code that other students have turned it, and ask them to explain how each function works, and then pick which one they find easiest to understand. This forces students to reflect on their code and hopefully think about how they can do better. One such homework is included in Appendix **??**.

As well as assessing high-level programming skills, some homeworks focus on lower-level skills. A certainly fluency in basic skills (data manipulation, writing functions, identifying errors) is necessary before they can be fluidly combined to solve bigger problems. To practice these skills we assign programming drills, made up of many simple problems. Each problem only requires a few minutes of thought, and stringing many together helps practice common techniques so that they can be quickly retrieved from memory. These drills are graded based on correctness with the assumption that most students will achieve grades of 90%+.

4 Conclusion

Teaching data analysis and programming is important, but hard, particularly when it comes to providing useful assessments. This paper has outlined some of the techniques that we have found to be most useful, and hopefully will encourage others to teach these important areas. Struggling to teach data analysis has also made us aware of how little literature there is on the topic, and we have made research in this area a priority.

A class that teaches data analysis and programming, if designed carefully, does not require extensive prerequisites (either statistical or computational). It provides an interesting introduction to statistics but it still forces students to struggle with some of the most important statistical and computational thinking. At Rice, the class has proven to be extremely popular, growing from 8 students per year to 60 students per semester in a little over three years. It has also contributed to the considerable growth of the statistics major.

References

George E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976. ISSN 01621459. doi: 10.2307/2286841. URL `http://dx.doi.org/10.2307/2286841`.

W. John Braun. An illustration of bootstrapping using video lottery terminal data. *Journal of Statistics Education*, 3(2), 1995. URL `http://www.amstat.org/publications/jse/v3n2/datasets.braun.html`.

Christopher Chatfield. *Problem Solving : A Statistician's Guide*. Chapman & Hall, 1995.

D. R. Cox. Applied statistics: A review. *Annals of Applied Statistics*, 1(1):1–16, 2007. URL `http://projecteuclid.org/DPubS?verb=Display&#38;version=1.0&#38;service=UI&#38;handle=euclid.aoas/1183143726&#38;page=record`.

Garrett Grolemund and Hadley Wickham. Dates and times made easy with lubridate. *Journal of Statistical Software*, 40(3):1–25, 2011. URL `http://www.jstatsoft.org/v40/i03/`.

Peter Huber. *Data Analysis What Can Be Learned from the Past 50 Years*. John Wiley Sons, Inc., Hoboken, New Jersey, 2011.

Brian W Kernighan and P J Plauger. *The Elements of Programming Style*. Computing Mcgraw-Hill, 1978.

Paul Murrell. Comparing non-identical objects. *R News*, 8(2):40–47, October 2008. URL `http://CRAN.R-project.org/doc/Rnews/`.

D. Nolan and D. Temple Lang. Computing in the statistics curricula. *The American Statistician*, 64(2):97–107, 2010.

Barbara Oakley, Richard M Felder, Rebecca Brent, and Imad Elhajj. Turning student groups into effective teams. *Journal of Student Centered Learning*, 2(1):9–34, 2004. URL `http://www4.ncsu.edu/unity/lockers/users/f/felder/public/Papers/Oakley-paper(JSCL).pdf`.

J. W. Tukey and M. B. Wilk. Data analysis and statistics: an expository overview. In *AFIPS '66 (Fall): Proceedings of the November 7-10, 1966, Fall Joint Computer Conference*, pages 695–709, New York, NY, USA, 1966. ACM. doi: http://doi.acm.org/10.1145/1464291.1464366.

Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL `http://www.jstatsoft.org/v21/i12/paper`.

Hadley Wickham. stringr: modern, consistent string processing. *R Journal*, 2(2):38–40, 2010a. URL `http://github.com/hadley/stringr`.

Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1): 3–28, 2010b.

Hadley Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL `http://www.jstatsoft.org/v40/i01/`.

CJ Wild and M. Pfannkuch. Statistical thinking in empirical enquiry. *International Statistical Review*, 67(3): 223–248, 1999.

A Syllabus

The syllabus is organised around the three main themes: visualisation, data manipulation, and programming. The course is arranged into eight modules each made up of three lectures and focussed on one theme. Each theme is covered multiple times, so each repetition can go into more depth and teach deeper skills. The first repetition gives them the essential skills for their first data analysis project, and provides the foundations for future repetitions: we first make them proficient, then effective, then sophisticated.

Each module is described in more depth below. Only two modules are focussed on visualisation, because it is used so extensively in all lectures.

- **Introduction to ggplot2**: The basics graphical tools of statistics: scatterplots, bar charts and histograms. How to add additional variables to a plot using aesthetics (like colour, shape or size) and conditioning. Techniques that make it easier to compare 1d distributions (frequency polygons), and to deal with overplotting on scatterplots (boxplots, smoothers, 2d binning).

  Focus: visualisation. Data: diamond prices.

- **Data frames**: subsetting (numeric, logical, character), data input and output, `.csv` vs. `.rdata`, missing values.

  Focus: data manipulation. Data: EPA fuel economy 1973-2009.

- **Writing functions**: basic strategy for functions (solve for specific case and then generalise) and basic components of a function (arguments, body, return value), control flow (if/if else/else, for loops).

  Focus: programming. Data: slot machine returns (Braun, 1995).

- **Data manipulation**: joining multiple data sets, reordering data frames, group-wise aggregation and transformation, the split-apply-combine strategy (Wickham, 2011).

  Focus: data manipulation. Data: top 1000 US baby names 1890-2008.

- **ggplot2 theory**: framework for critiquing graphics, layered grammar of graphics (Wickham, 2010b), scales and themes for tweaking graphics for communication rather than exploration.

  Focus: visualisation. Data: US airways on time arrivals.

- **Special data**: working dates (Grolemund and Wickham, 2011), strings (Wickham, 2010a), and regular expressions. Particularly focus on the different between a string and it's representation in R (why a regular expression to replace a single backslash is \\\\).

  Focus: programming. Data: emails.

- **Data cleaning**: the basic data structures, when to use each, the the 4 Cs of clean data, the difference between molten, long and wide data, and changing between them (Wickham, 2007).

  Focus: data manipulation. Data: billboard top 100 songs 1957-2008.

- **Advanced programming**: debugging, optimisation, vectorisation, and why floating point math is not the same as pure math.

  Focus: programming.

An additional module covers professional development, including team work, report writing, and poster presentations, and is spread over the entire semester.

B Code review

For homework, students are asked to read the following three functions which compute the prize awarded when given the windows displayed by a slot machine. After reading the code, they write one page summary describing the basics of how each function works, which one they liked best, and what they learned about writing clear code.

```
prize1 <- function(slots) {

  if (!is.character(slots) || length(slots) != 3) {
    stop("Slots should be a character vector of length 3.")
  }
  same <- unique(slots)

  if (length(same) == 1) {    # All the same
    prizes <- c("DD" = 800, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
    return(unname(prizes[same]))
  }

  ds <- sum(slots == "DD")

  if (length(same) == 2 & ds > 0) {    # All the same with wilds
    same = setdiff(same, 'DD')
    prizes <- c("DD" = 800, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
    return(2^ds * unname(prizes[same]))
  }

  if (all(same %in% c("B", "BB", "BBB","DD"))) return(2^ds * 5)  # All bars
  cs <- sum(slots == "C")

  if(cs > 0){
    if(cs == 1){
      if(ds == 0) return(2)  # 1 carat, no diamonds
      if(ds == 1) return(10)
    }

    if(cs == 2){
      return(5)
    }
  }

  return(0)
}

prize2 <- function(slots) {
  if (!is.character(slots) || length(slots) != 3) {
    stop("slots should be a character vector of length 3")
  }
  same <- unique(slots)
  a <- c("7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
  ndd <- sum(slots == "DD") # number of diamonds
  nb <- sum(slots %in% c("B", "BB", "BBB")) # number of bars
  nc <- sum(slots == "C") # number of cherries
  if (ndd == 3) {
    # Three diamonds
    800
  } else if (ndd == 2) {
    # Two diamonds
    prizes <- a * 4
    unname(prizes[slots[slots != "DD"]])
  } else if (ndd == 1) {
    # One diamond
    if (length(same) == 2) {
      # One diamond with the other same windows
      prizes <- a * 2
      unname(prizes[same[same != "DD"]])
    } else if (nb == 2 || nc == 1) {
      # One diamond with two different bars or with one cherry and one zero
      10
    } else {
      0
    }
  } else if (length(same) == 1) {
    # All the same
    prizes <- a
    unname(prizes[same])
  } else if (nb == 3) {
    # All bars
    5
  } else {
    # Cherries
    c(0, 2, 5)[nc + 1]
  }
}


# find the prize of a given window combination
prize3 <- function(slots) {
  if (!is.character(slots) || length(slots) != 3) {
    stop("slots should be a character vector of length 3")
  }
  same <- unique(slots)

  if (length(same) == 1) {
    # All the same
    prizes <- c("DD" = 800, "7" = 80, "BBB" = 40, "BB" = 25, "B" = 10,
      "C" = 10, "0" = 0)
    unname(prizes[same])

  } else if ((length(same) == 2) & ("DD" %in% slots) & (same[1] != "DD")) {
    # All the same with one or two DD’s, the first entry in same is not DD
    ds <- sum(slots == "DD")
    prizes2 <- c("7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
    unname(prizes2[same[1]]) * c(1, 2, 4)[ds + 1]

  } else if ((length(same) == 2) & ("DD" %in% slots) & (same[1] == "DD")) {
    # All the same with one or two DD’s, the first entry in same is not DD
    ds <- sum(slots == "DD")
    prizes2 <- c("7" = 80, "BBB" = 40, "BB" = 25, "B" = 10, "C" = 10, "0" = 0)
    unname(prizes2[same[2]]) * c(1, 2, 4)[ds + 1]

  } else if (all(same %in% c("B", "BB", "BBB", "DD"))) {
    # All bars
    ds <- sum(slots == "DD")
    5 * c(1, 2, 4)[ds + 1]

  } else if ("C" %in% slots) {
    # Diamonds and cherries
    cs <- sum(slots == "C")
    ds <- sum(slots == "DD")

    c(0, 2, 5)[cs + ds + 1] * c(1, 2, 4)[ds + 1]

  } else {
    0
  }
}

# The function takes an inputed string from slot machine data
# and outputs the expected prize.  Diamonds are wild and
# the presence of diamonds double the total prize per diamond.
# Prizes determined by 3 of the same, all bars, or cherries.
prize4 <- function(windows) {

  payoffs <- c("DD" = 800, "7" = 80, "BBB" = 40,
    "BB" = 25, "B" = 10, "C" = 10, "0" = 0)

  same <- length(unique(windows)) == 1

  diamond_wild <- length(unique(windows)) == 2 &
    any(windows %in% "DD")

  allbars <- all(windows %in% c("B", "BB", "BBB", "DD"))

  diamonds <- sum(windows == "DD")

# Determining Prizes---------------------------------------------
  if (diamonds == 3){
    800
  } else if (same | diamond_wild) {
    payoffs[windows[!(windows %in% "DD")][1]] *
    c(1, 2, 4)[diamonds + 1]
  } else if (allbars) {
    5 * c(1, 2, 4)[diamonds + 1]
  } else {
    cherries <- sum(windows == "C") + sum(windows == "DD") *
    as.numeric(any(windows %in% "C"))

    c(0, 2, 5)[cherries + 1] *
    c(1, 2, 4)[diamonds + 1]
  }
}
```

C Rubrics

The following two pages include our rubrics for grading data analysis homeworks and code.

	Outstanding (A+)	Good (A)	Acceptable (B)	Needs work (C)	Inadequate (F)
	5	**4**	**3**	**2**	**1**
Curiosity	Intense exploration and evidence of many trials and failures. You have looked at the data in many different ways before coming to your final answer. You have gone beyond what was asked: additional research from other sources used to help understand/explain findings. Your explanation and presentation is creative	Plenty of exploration and investigation. Some additional research helps explain findings, and some of your ideas are creatively presented and explained.	Some exploration, but little evidence that you have selected the best of many ideas. Little or no additional research.	You have done the bare minimum that was asked. There is no evidence to suggest that you tried multiple approaches (tables, graphics, or models) before coming to your final conclusion.	Questions are simple, and there is no evidence of exploration. You have not come up with your own questions of the data, but relied on those we discussed in class
Scepticism	You suggest multiple explanations for a given finding, and use multiple tools to explore surprising results. You present one or two as the most plausible, but have allowed for the possibility that you are wrong. You are self-critical: What did I do well? What did I do poorly? What have I missed? How could I do better next time? You identify flaws in methodology and provide suggestions as to how they could be remedied You don't blindly accept perceived wisdom, but challenge preconceived notions and come up with interesting new ways of testing them.	You are sceptical and self-critical, but not consistently. There is some critical analysis, and some use of multiple techniques to answer the same question.	You haven't blinded accepted findings, but you haven't come up with many ways to check your results either. There is little self-criticism and little evidence to suggest you have thought about how to do better in the future.	Some findings accepted without question. Self-criticism weak.	Findings accepted uncritically. Leaps of logic without justification. You have not thought about how to do better next time.
Organisation	Findings very well organised. Clear headings demarcate separate sections. Excellent flow from one section to the next. The paper is easy to scan. An abstract or summary at the start of the paper briefly summarises your approach and findings. Conclusions at the end present further questions and ways to investigate more. Tables and graphics carefully tuned and placed for desired purpose.	Findings well organised and sections clearly separated, but flow is lacking. Each section has clear purpose. Tables and graphics clear and well chosen	Generally well organised, but some sections muddled. Tables or graphics appropriate, but some are poorly presented - too many decimal places, poorly chosen aspect ratio etc.	Sections unclear and no attempt to flow from one topic to the next. Graphics and tables poorly chosen to support questions. Some have fundamental flaw.s	It is hard to read your paper. There are no headings, figures are far away from where they are referenced in the text. There is no summary or conclusion.

	Outstanding (A+)	Good (A)	Acceptable (A-/B+)	Needs work (B/C)	Inadequate (F)
	5	**4**	**3**	**2**	**1**
Planning	Introductory comment describes overall strategy and gives evidence of preliminary planning. Thoughtful problem decomposition breaks the problem into independent pieces that can be solved easily.	Evidence of planning before coding, but some flaws in overall strategy.	More planning needed: overall strategy ok, but have missed some obvious ways of making the code simpler.	It all hangs together, but planning was absent or rushed.	No evidence of planning. Strategy deeply flawed.
Execution	Mastery of R vocabulary means that the absolute minimum amount of code is used to get the job done. Code free from duplication. Each function encapsulates a single task, and repeated tasks are performed by functions, not copy and paste	Workable, but not elegant. Common programming idioms used to reduce code. For example: character subsetting instead of complicated if statements; vectorised functions instead of for-loops.			Functions used inapproriately, or existing functions reinvented. Extensive use of copy and paste.
Clarity	Code is a pleasure to read, and easy to understand. Code and comments form part of a seamless whole.	Comments used to discuss the why, and not how of code; to provide insight into complicated algorithms; and to indicate purpose of function (if not obvious from its name). Comment headings used to separate important sections of the code. Variables, functions and arguments named concisely but descriptively.	Generally easy to read, but some comments used inappropriately: either too many, or too few. Some variable names confusing.	Hard to understand. Poor choice of names and comments do not generally aid understanding.	Cannot understand code. If it works, I have no idea why.

One point will be deducted for each of the following style guide violations:
file naming, identifier naming, spacing, curly braces, indentation, line length, assignment