A Case Where It Is Better to have an Unstandardized measure of the Right Construct than a Standardized Measure of a Related One: Application to Coding Interviews Within a Course in SAS Programming

Gregory Samsa[1,*], Megan Neely[1], Steven Grambow[1], Marissa Ashner[1], Gina-Maria Pomann[1], Laura Coutts[1] & Jesse Troy[1]

[1]Department of Biostatistics and Bioinformatics, Duke University School of Medicine, United States

*Correspondence: Department of Biostatistics and Bioinformatics, Duke University School of Medicine, United States. E-mail: greg.samsa@duke.edu

Received: September 5, 2025 Accepted: October 15, 2025 Online Published: October 24, 2025

doi:10.5430/jct.v14n4p163 URL: https://doi.org/10.5430/jct.v14n4p163

Abstract

Interview-based examinations provide richer data than formats such as multiple choice and short answer, albeit at the cost of being less standardized. This describes administering a coding interview as the final examination in a class on SAS programming, plus primarily qualitative reflections. We conclude that when the goal is to assess facility with programming an interview-based examination should come into especial consideration.

We argue that a coding interview measures the right thing, namely how well the student designs and writes SAS programs – which in turn depends on factors such as general programming literacy, critical SAS-specific knowledge, ability to design SAS programs, and the ability to engage in problem solving as part of the process of program development -- rather than something that is merely correlated with this core construct, as would be the case for the objective questions that are included within typical certification tests. In doing so it is not completely standardized, but sufficiently so. This examination format more closely matches how students engage in SAS programming in actual practice: for example, by incorporating web searching. Moreover, it has the innovative and desirable property of embedding instruction in addition to evaluation. Coding interviews are a time-intensive form of evaluation, but much more is learned about student performance, there is an opportunity to teach as you go, and the time is well spent.

Keywords: coding interviews, SAS programming, student assessment

1. Introduction

In contradistinction to traditional formats such as multiple choice and short answer, interview-based examinations have several advantages. These advantages are discussed later.

However, a potential disadvantage of interview-based examinations is that they cannot be fully standardized. Because students are not asked identical questions an interview-based examination might be considered "not reproducible" and thus "unfair" and “subject to challenge". Indeed, in our experience this perception prevents some instructors from being willing to consider using interviews. While we acknowledge that this is a reasonable concern, we argue that it need not be a decisive one.

This is a qualitative reflection on our experience, accumulated over multiple years, in administering a coding interview as the final examination in a class on SAS programming (SAS, 2025) within a Master of Biostatistics program (Neely *et al*, 2018; Samsa, 2018a; Samsa *et* al, 2018b; Samsa, 2020; Samsa, 2021; Troy *et al,* 2021; Troy *et al,* 2022a; Troy *et al*, 2022b; Troy *et al*, 2022c; Troy *et al*, 2023; Troy *et al*, 2024; Troy *et al*, 2025) . This class has been described elsewhere (Samsa, 2020; Troy *et al*, 2023). Briefly, its primary goal is to teach basic programming concepts such as literate programming, program design, algorithms and data structures, with SAS (SAS Institute, 2011) serving as the use case. In other words, the goal is to teach students to become excellent programmers in general and excellent SAS programmers in particular. We ultimately conclude that when the instructor’s goal is to assess facility with programming an interview-based examination should come into especial consideration.

2. Methods

2.1 Goals

The goals of the coding interview are based on our experience in performing interviews as part of the hiring process (e.g., for software engineering and biostatistical staff positions). Indeed, they are intended to be consistent with guidelines for using coding interviews for this purpose (Behroozi, *et al*, 2019). In an actual coding interview, the interviewer attempts to assess the related constructs of (1) the interviewee's overall skill as a programmer; and (2) their skills as a SAS programmer. Overall mastery of programming is a multi-faceted construct, and includes computational thinking, facility with algorithms and data structures, and the use of literate and reproducible programming practices (Knuth, 1984; Denning *et al*, 1989).

Our rationale for assessing overall mastery of programming in addition to SAS-specific skills is that it is not only important for lifelong learning but key to success on the job. In other words, on the job general facility with programming (and other traits such as curiosity, critical thinking and organization) provides the tools to rapidly learn new skills, including those that pertain to SAS.

Our assessment is intended to be consistent with how students will use SAS on the job, where details about SAS syntax can be discovered by web searching, generative artificial intelligence (AI), etc. -- if a student understands enough about the structure of SAS, they truly can teach themselves. Accordingly, we focus less on the details of SAS syntax than would a certification-based examination.

2.2 Constructs

The examination focuses on four constructs: (1) general programming literacy; (2) critical SAS-specific knowledge; (3) ability to design SAS programs; and (4) ability to engage in effective problem solving as part of the process of program development. The second construct measures “what students know”, whereas the other constructs measure “how students think”. These latter constructs are consistent with a constructivist philosophy of education, in that (1) we directly observe what students do; and (2) we attempt to extract information about the soundness of the mental model that students have constructed around the discipline of programming (Fosnot, 1996; Phillips, 1995).

Simple examples of general programming sophistication are the ability to use "do loops" and "arrays", which are common program structures regardless of language.

An example of critical SAS-specific knowledge includes recognizing that SAS automatically creates variables during the "DATA step" -- for example, it creates an indicator variable tracking whether the observation is the first one in a group defined using "BY processing".

The ability to design SAS programs is based on using programming algorithms that will perform well in SAS. For example, one way to determine that an id variable is unique is to read the input dataset "BY id" and then check whether the values of the automatic variables "FIRST.id" and "LAST.id" both equal 1 (i.e., the Boolean value for "true").

In this context, effective problem solving includes techniques for testing SAS programs one step at a time (i.e., "unit testing"), debugging programs (e.g., using the SAS log to identify errors), and using web searching to find model code when the student has strayed beyond the limits of their current knowledge base and is thus uncertain what to do next. Indeed, coding interviews in other contexts often include tasks of increasing difficulty with the intention of discovering a situation where the student doesn't know what to do (Troy *et al*, 2023). There, what matters less is where the limit of their knowledge base is encountered (within reason) and what matters more is what they do after they reach that limit. This examination follows a similar philosophy: the meta-construct to assess is how well the student can design and write a SAS program (broadly defined), problem solving is a key component of program writing, and being able to effectively utilize external resources is a key component of problem solving.

2.3 Questions

The examination contains questions about general topics, data management topics, statistical topics, report-related topics, and advanced topics. Appendix A contains examples of each category of question. Questions that are code-related begin with "*What does the following code do?*" rather than “*How would you write code to accomplish this task?*”. This is a departure from typical interview practice that not only helps the student to get off to a good start, it also directly addresses algorithms, data structures, and basic syntax (see appendix B). In our view, this not only represents an innovative way to perform this portion of a coding interview, but one which is directly linked with the interview's evaluative goals.

2.4 Reproducibility

Some elements of reproducibility can be built into a coding interview -- for example, with standardized starter questions, selecting a minimum number of questions from each category, using a grading rubric, recording the interview to facilitate regrading and quality improvement, etc. These elements are not intended to create identical examinations for every student but nevertheless do serve to increase reproducibility.

2.5 Distinguishing Features

Although some coding interviews are confrontational, this examination is structured in a way that is intended to put the student at ease (and, thus, do their best work). As noted above, instead of asking students to write code *de novo*, they are provided with a place to start, asked to explain that code, and then expand upon it. This structure allows the instructor to intervene and explain any component of the code that is misunderstood. Thus, the coding interview potentially embeds an element of instruction in addition to evaluation.

Another distinguishing feature is that web searching is allowed. In our experience, apart from being consistent with how students will write SAS programs in actual practice, this helps differentiate between students who have and have not been completing the programming assignments on their own during the semester. Those who have been actively engaged with the course material can generate a reasonable plan for their program -- for example, by describing its algorithm in pseudocode -- but might struggle with the details of the SAS syntax for implementing that algorithm. Such students can effectively utilize the results of their web searches to fill in the missing details. Students who haven't been completing programming assignments on their own tend to be unable to effectively utilize search results.

Yet another distinguishing feature is that questions can be targeted toward the individual characteristics of students. This is discussed later.

2.6 Preparation

During the semester the instructor performs a simulated coding interview with another faculty member. After each question is completed, both parties describe their thinking: for example, the interviewer describes what they were looking for when they asked the question, and the interviewee describes their strategy for answering the question and demonstrating their facility with SAS. Moreover, many of the class assignments involve describing SAS code in plain English, making modest extensions to basic SAS code, etc. -- in other words, they provide practice in what students are asked to do during their coding interview.

2.7 Example Examination

Appendix B illustrates the flow of typical examination, plus commentary.

2.8 Grading Rubric

Although they are conceptualized as being separate, in practice the constructs being evaluated are interrelated. For example, consider an excellent programmer with experience in multiple programming languages including SAS. Part of general programming literacy pertains to program design – it is performed systematically, sequentially, and uses modularized building blocks that are tested along the way. General programming literacy includes attention to data structures so, for example, the concept of a SAS dataset as a rectangular array of data plus separate descriptors (i.e., meta-documentation) is straightforward, as is the notion of designing algorithms that take advantage of how SAS is structured (e.g., as a default, SAS datasets are created one row at a time, from top to bottom). An excellent general programmer will have a sound plan for their code, will learn most SAS syntax quickly and will recall the basics (at least). They will know what they don’t know and be able to effectively use external resources to augment their knowledge.

Given the above, our grading rubric doesn’t evaluate each construct separately but instead provides a general description such as the above and assesses how closely a student approaches this ideal. Of note: students can be graded as “outstanding” even if they need external assistance on details of SAS syntax, so long as they can use this assistance effectively and the assistance is with “details” rather than more fundamental concepts such as the overall structure of SAS and the ability to design sound programming algorithms consistent with that structure.

In practice, those students who struggle do so on multiple elements of the above, and so it is satisfactory to base grades on how closely the above ideal is achieved (e.g., very close to the ideal, relatively close to the ideal but with modest deficiencies, far from the ideal but making progress, far from the ideal). Struggling students almost always lack general programming literacy, which is then manifested in multiple ways as they attempt to use SAS.

Feedback to students is provided in terms of how closely they approached the above ideal. For example: "*You were able to describe how the starter SAS programs worked and were able to propose how to extend them to solve more complex problems. You struggled to find the SAS syntax to do so, didn't use wonderful search terms as you tried to research what to do next, but once the instructor suggested better search terms you were able to use the results to generate the required syntax. It was encouraging to see that you printed intermediate values to diagram how your SAS program operated and your overall approach to program development wasn't fully systematic but nevertheless was reasonable. My overall assessment is that your general programming skills are good but not outstanding, your ability to use SAS is consistent with that, and what's needed is (1) approaching program development more systematically; and (2) more practice with SAS. My overall assessment is that you haven't yet approached the ideal, but also that your deficiencies are relatively modest and easily fixable."* Providing feedback in this format (which is also framed to be as encouraging as is realistically feasible) might have served to discourage challenges about grades.

3. Results

3.1 Student Evaluations

The median student evaluation of the overall course (n=10 respondents) was "*very satisfied*". The median responses to "*I can find and use help to learn additional SAS content*", "*I can design a SAS program*" and "*If a program design has been prepared, I can write SAS code*" was "*very confident*".

Less formally, after each coding interview students (n=23) were asked how closely the experience aligned with their expectations. All reported that it was consistent with both their expectations and the simulated coding interview with the instructor and their colleague. Students also reported that the interview was a fair assessment of their ability to program in SAS. Multiple students provided the unprompted response that being able to perform web searching as part of the interview was helpful.

3.2 An Alternative Type of Examination

Our coding interview can be contrasted with a typical standardized examination. An illustration of a standardized examination for SAS programming is provided by a study guide for a SAS certification examination published by the SAS Institute (SAS Institute, 2024). All questions are multiple choice. One type of question pertains to the details of SAS syntax to accomplish a small task*:* for example, *"Which response prints the date variable DATE as '01JAN2024'?".* Another type of question pertains to precisely how SAS parses a small unit of code: for example, "*The highlighted code will create a SAS dataset named TEMP1. Which answer correctly lists the contents of TEMP1?"*. Yet another type of question pertains to how specific statements are used: for example, "*Which response best describes what a RETAIN statement accomplishes?"*.

In an actual certification examination, all students are asked the same questions. Grading is rapid and automated, and all scores at or above a threshold are passing. In a classroom setting multiple threshold values can be used: for example, 98-100=A+, 95-97=A, etc. This type of examination has various advantages – essentially, that students are asked identical questions that are objective. The disadvantages follow from measuring what can be objectively quantified, not necessarily what is most important.

3.3 Personal Experience

The limitations of traditional testing can be illustrated by the personal experience of the first author. Back in the day, the "theory" component of the qualifying examination in my doctoral program was a 4-hour examination. The format was closed book, although 10 pages of notes could be used, thus allowing the designers to presume that the details of specific formulae would be common knowledge. The internet did not yet exist.

My fellow students had discovered that the examination questions weren't developed *de novo* but were instead copied from illustrations from five of the main textbooks on the theory of mathematical statistics. Having weaker mathematical training than my peers, I spent the summer before the examination hand-copying all the relevant illustrations and thus creating 160 pages of notes. These notes were then photocopied and reduced to 10 pages in very small font, which I brought to the examination along with a magnifying glass. The examination was completed in approximately 15 minutes, all devoted to the mechanics of writing, and the score was 100%.

The graders wrongly assumed that I had mastered the underlying construct which the test was intended to assess – namely, the ability to use the same techniques in a new context such as writing a dissertation. On reflection, close exposure to the content resulted in some degree of learning. I did actively engage with the material and could

reproduce many of the derivations, although only for a limited amount of time -- within a few weeks all that remained was a general appreciation for the structure of this body of knowledge. I could barely, if at all, apply the contents of the examination to create new knowledge, and wisely selected a dissertation topic that heavily relied on simulation instead of mathematics.

On reflection, the pedagogic premise of this examination was similar to that of a credentialing test: namely, that during the time that a student makes the effort to learn / memorize a large number of technical details that would be encountered when they practice the discipline then they, *inter alia*, will also master the discipline -- thus, testing recall / use of these details is equivalent to objectively testing mastery of the discipline. In fact, these two constructs are correlated but different. Here, the technical details were temporarily memorized, but the underlying concepts weren't mastered to the desired level of proficiency. To which a lifetime of previous coursework can be blamed: insufficient mathematical background is not at all trivial to remediate -- at best, this requires concerted effort over more than a summer.

4. Discussion

4.1 Summary

We have described the use of a coding interview to evaluate student performance in a class on SAS programming. Its structure is closer to coding interviews for software engineers and professional programmers than to the standardized evaluations typically used in certification. Its design is ultimately derived from the desire to evaluate (1) how well students design and "write SAS programs; and (2) core constructs that contribute to doing so (i.e., general programming literacy, critical SAS-specific knowledge, ability to design SAS programs, ability to engage in effective problem solving). Evaluations from both students and the instructor were encouraging.

4.2 Personalization

In an academic context where grades require justification, a coding interview must strike a balance between personalization and consistency. Some elements of consistency include explicit statement of interview goals, a rubric for grading, drawing from a bank of initial questions (although follow-up questions will uniquely depend on how students respond), and evaluating the same core constructs for all students.

One element of personalization pertains to the choice of questions, especially the follow-up queries in response to initial answers. As an example, the most recent class included students who aren't training to become statisticians, and the statistics-related questions were modified for them. More specifically, questions focused on creating straightforward data summaries (e.g., frequency distributions) and capturing them for further analysis, a skill that is relevant to non-statistical investigators. Follow-up questions focused on graphical summaries used by non-statistical investigators but not on the details of coding complex statistical analyses.

We argue that, when used properly, the ability to ask students different questions is more a positive feature than a source of inconsistency. For example, only advanced students need be asked the advanced questions -- for other students, what they do when faced with a problem outside the limits of their knowledge will already have been discovered. In practice, beginning with general questions also serves to screen students about how they conceptualize programming -- for example, as software engineers, as users, etc. The choice and phrasing of subsequent questions can be consistent with that conceptualization.

Here, the general pedagogic principle is that while the same things should be evaluated for all students using the same criteria for mastery they need not be evaluated identically. Indeed, personalizing the evaluation allows students to utilize their unique strengths. The same principle applies to students who require special types of accommodation.

Another element of personalization pertains to allowing students to use web searching when the limits of their knowledge base are exceeded. Search topics will differ from student to student, depending on where they reach the limit of their knowledge. Effective use of web searching is part of problem solving, which is one of the components of effective programming that the coding interview evaluates.

Another element of personalization is the ability for both students and the instructor to ask clarifying questions. Traditional standardized testing risks miscommunication in both directions -- the student might not fully understand the question, and the instructor might not fully understand the answer. A traditional test doesn't provide the opportunity to either discover or fix this.

4.3 Role of Interview as a Teaching Tool

A coding interview offers the opportunity for the instructor to ask struggling students follow-up questions in the hope

of discovering the precise nature of their deficiencies. Ideally, the source of these deficiencies can be addressed during the interview.

A noteworthy element of a coding interview is the ability to use the examination not just to evaluate but also to teach. Instruction can occur at multiple points during the interview: for example, (1) when the student explains the model code provided at the start of a problem; (2) as the student designs an algorithm to extend the model code; and (3) as the student attempts to interpret the results of their web search and apply that information to the task at hand.

4.4 Cheating

Yet another noteworthy element is that this type of examination can be defeated by neither memorization nor cheating, providing students with a strong incentive to learn the material. Which is fortunate: memorizing a list of facts doesn't transform students into effective programmers. We know of no way to cheat on an interview exam, which saves the time and effort on the part of the instructor that would otherwise be devoted to dealing with this unpleasant topic. For example, rather than engaging in the Sisyphean task of trying to keep track of how students are currently using generative AI, we simply allow its use, directly observe the results, and assess the degree to which the student can apply them to the task at hand. Indeed, this suggests an additional type of question, quite relevant to eventual success on the job: namely, "*This SAS program, written by generative AI, was intended to accomplish task X – explain what is wrong and how to fix it*".

4.5 Student Response

Although this form of evaluation was generally unfamiliar to students, 100% reported that the interview was essentially as they expected and that it adequately assessed their facility with SAS programming. Over the years, although a few students have requested clarification about the grading rubric (which is described in advance of the examination) no student has ever challenged their grade, which we believe helps address potential concerns about fairness. We posit that this is because their level of mastery (or lack thereof) became clear to them during the interview.

4.6 Comparison with Credential-Based Examinations

We do not suggest that standardized tests are a uniformly poor approach as they are an excellent tool for a large class of evaluations. In the present context, there is a small core of SAS syntax that (1) must be applied when writing a typical SAS program; and (2) will have been encountered sufficiently often as to be naturally committed to memory. Utilizing a standardized evaluation of this information not only serves as a check on basic knowledge but also verifies that students have been actively engaged with the coding assignments that contribute to the course grade. In other words, this small core of critical SAS syntax is both crucial to evaluate and well-measured using a standardized test such as a credentialing examination.

On the other hand, we argue that how well students can design and write a SAS program is a construct that is too broad to be effectively covered by a standardized evaluation. For this purpose, recall of facts correlates with the overall construct, but insufficiently well to serve as the sole basis for evaluation.

This coding interview is based on a different premise than a typical standardized (e.g., credential-based) test -- a premise that is consistent with that of coding interviews provided to software engineers -- namely, that the interview should focus on program development and problem solving, including problem solving beyond the limits of the student's knowledge base. In other words, a coding interview attempts to accomplish a fundamentally different and "bigger-picture" goal -- namely, discovering how well a student will design and write SAS programs. Facility with technical details is part of excellent SAS programming, and is assessed during the coding interview, but is not the most important part, if for no other reason than that technical details are easily discoverable through web searching, generative AI and similar resources. The coding interview assesses how well students write SAS programs by directly observing how they do so.

4.7 Limitations of Interview-Based Examinations

Despite their positive characteristics, two limitations are embedded within essentially all interview-based examinations: (1) standardization; and (2) instructor time / effort. Standardization has been previously discussed. An issue with instructor time is scalability: for example, if it takes X hours to create a thoughtful standardized examination for 10 students then amount of time per student is X/10. To give the same test to 100 students the preparation time per student drops to X/100. This figure doesn't account for the additional follow-up related to grade challenges, dealing with cheating, etc., but is in the correct order of magnitude. Moreover, examination questions can be reused (although this is often discovered by students, thus reducing their effectiveness). On the other hand, if it

takes Y hours per student to perform an interview the amount of time per student will be Y, regardless. If the class is sufficiently large performing the interviews – which typically lasted for 30-45 minutes in this case -- might not be feasible. Also, it is helpful to develop a reasonably large number of questions within each category from which to select, as this helps the instructor remain engaged and thus ensures that the last few interviews receive the same level of attention as the first.

The additional (and ongoing) effort required to perform interviews begs the question of when, in general, this effort is sufficiently worthwhile. We argue that a necessary condition is that the goal is to assess how a student thinks about the content rather than the specific facts that they can recall. For coding, a precedent has already been set by job interviews that resemble the coding interview described here. Their rationale is (1) this measures adaptive thinking, which is a crucial job skill; and (2) command of specific facts can be assessed using standardized credentialing tests. Here, an additional benefit is that students receive practice in the sort of interviews that they are likely to encounter during their job searches. Moreover, our program places a heavy emphasis on communication skills, and the interview provides an opportunity to demonstrate those skills.

An additional necessary condition is that the instructor has adequate time to perform the interviews, which in turn depends on class size and interview length. In our experience, it is preferable to perform relatively few interviews per day and stretch out the number of days. If an incentive can be offered for some of the students to sign up early, so much the better.

4.8 Additional Limitations

A limitation of this version of a coding interview is the lack of the validation process that is typical of teaching methods that are more fully developed. For example, some elements of that process could include more formal specifications for how the interview is performed, additional development of the grading rubric, agreement studies where interviews are recorded and then graded by multiple raters, how well interview results correlate with other measures of programming performance (e.g., in class work, eventually: on the job), etc. This current report isn't intended as a description of a fully validated instrument but, instead, as proof of concept describing a type of evaluation that is promising but uncommon.

5. Conclusion

In conclusion, we argue that a coding interview measures the right thing -- namely how well the student designs and writes SAS programs (broadly conceptualized) -- rather than something that is quantified, somewhat correlated with the right thing, but different. In doing so it is not completely standardized, but sufficiently so. This examination format allows more closely matches how students perform SAS programming in actual practice: for example, by incorporating web searching and generative AI. Moreover, it has the innovative and desirable property of embedding instruction in addition to evaluation.

Nowadays students realize that an answer (although not necessarily a fully correct one) to most questions can be found online. They risk becoming adept at discovering answers that others (whether human or AI) provide rather than learning how to critically think about their discipline and, thus, be able to provide their own answers to new questions. The ability to uncritically search for answers provided by others isn't a skill that is strongly valued by employers, although students don't necessarily realize this until they enter the workforce. Take-home examinations decreasingly represent students' own work. Standardized examinations require increasingly heavy proctoring to prevent electronically enhanced cheating and, for statistical programming, don't assess what is most important. Ideally, an examination format measures what is most important, is impervious to cheating, and requires students to think for themselves. Interviews meet all these criteria.

References

Behroozi, M., Parnin, C., & Barik, T. (2019). Hiring is Broken: What Do Developers Say about Technical Interviews? *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2019-October, 15-23. https://doi.org/10.1109/VLHCC.2019.8818836.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Computer*, *22*(2), 63-70. https://doi.org/10.1109/2.19833

Fosnot, C. T. (1996). *Constructivism: Theory, Perspectives, and Practice.* New York, New York: Teachers College Press.

Knuth, D. E. (1984). Literate programming. *Computer Journal*, *27*(2). https://doi.org/10.1093/comjnl/27.2.97

Neely, M. L., Troy, J. D., Gschwind, G., Pomann, G.-M., Grambow, S. C., & Samsa, G. P. (2022). Preorientation Curriculum: An Approach for Preparing Students with Heterogenous Backgrounds for Training in a Master of Biostatistics Program. *Journal of Curriculum and Teaching, 11(*4), 120-138. https://doi.org/10.5430/jct.v11n4p120

Phillips, D. C. (1995). The good, the bad, and the ugly: The many faces of constructivism. *Educational Researcher*, *24*(7), 5-12. https://doi.org/10.3102/0013189X024007005

Samsa, G. (2021). Evolution of a Qualifying Examination from a Timed Closed-Book Format to an Open-Book Collaborative Take-Home Format: A Case Study and Commentary. *Journal of Curriculum and Teaching*, *10*(1), 47-55. https://doi.org/10.5430/jct.v10n1p47

Samsa, G. P. (2018a). A Day in the Professional Life of a Collaborative Biostatistician Deconstructed: Implications for Curriculum Design. *Journal of Curriculum and Teaching*, *7*(1), 20-31 https://doi.org/10.5430/jct.v7n1p20

Samsa, G. P. (2020). Using Coding Interviews as an Organizational and Evaluative Framework for a Graduate Course in Programming. *Journal of Curriculum and Teaching*, *9*(3), 107-140.

Samsa, G. P., LeBlanc, T. W., Locke, S. C., Troy, J. D., & Pomann, G.-M. (2018b). A Model of Cross-Disciplinary Communication for Collaborative Statisticians: Implications for Curriculum Design. *Journal of Curriculum and Teaching*, *7*(2), 1-11. https://doi.org/10.5430/jct.v7n2p1

SAS Institute Inc. (2011). *SAS Certification Prep Guide: Base Programming for SAS9* (3rd ed.). Cary, NC: SAS Institute Inc.

Troy, J. D., Granek, J., Samsa, G. P., Pomann, G.-M., Updike, S., Grambow, S. C., & Neely, M. L. (2022a). A Course in Biology and Communication Skills for Master of Biostatistics Students. *Journal of Curriculum and Teaching*, *11*(4), 120-138. https://doi.org/10.5430/jct.v11n4p120.

Troy, J. D., Neely, M. L., Grambow, S. C., Pomann, G. M., & Samsa, G. P. (2024). A curriculum review of programming courses in a Master of Biostatistics Program. *Journal of Curriculum and Teaching*, *13*(1), 405-420. https://doi.org/10.5430/jct.v13n1p405

Troy, J. D., Neely, M. L., Pomann, G. M., Grambow, S. C., & Samsa, G. P. (2022c). Administrative Considerations Pertaining to the Use of Creative Methods of Student Assessment: A Theoretically Grounded Reflection from a Master of Biostatistics Program. *Journal of Curriculum and Teaching*, *11*(5), 105-113. https://doi.org/10.5430/JCT.V11N5P105

Troy, J., Grambow, S. C., Neely, M. L., Pomann, G. M., Davenport, C., Ashner, M., & Samsa, G. (2025). Rationalizing the Mathematics Requirements for a Master of Biostatistics Program: A Case Study and Commentary. *Advances in Social Sciences Research Journal*, *12*(9), 215-222. https://doi.org/10.14738/assrj.1209.19397.

Troy, J. D., Pomann, G. M., Neely, M. L., Grambow, S. C., & Samsa, G. P. (2023). Are simulated coding interviews a fair and practical examination format for non-professional programmers enrolled in a master's degree program in biostatistics? *Journal of Curriculum and Teaching*, *12*(6), 253-264. https://doi.org/10.5430/jct.v12n6p253

Troy., J. D., McCormack, K. D., Grambow, S. C., Pomann, G. M., & Samsa, G. P. (2022b). Redesign of a first-year theory course sequence in biostatistics. *Journal of Curriculum and Teaching*, *11*(8), 1-12. https://doi.org/10.5430/jct.v11n8p1

Troy., J. D., Neely, M. L., Grambow., S. C., & Samsa., G. P. (2021). The Biomedical Research Pyramid: A model for the practice of biostatistics. *Journal of Curriculum and Teaching*, *10*(1), 10-17. https://doi.org/10.5430/jct.v10n1p10

Appendix A

Example questions

Two general questions are:

> *Describe some of the characteristics of SAS. If you like, you can compare SAS to R, Python, or some other programming language.*

Describe some excellent programming practices, and how you implement them in SAS.

The general questions are intended to assess big-picture understanding of programming in general and of SAS in particular. Many students choose to compare SAS to R, which is the topic of a programming course offered during the previous semester. All students are asked the same set of general questions.

A data management question is:

The program fragment below illustrates how to horizontally merge (join) two datasets. How does it operate? How would you modify the code to filter in records from both input files?

```
data both;
 merge one
       two;
 by id;

run;

proc print data=both;
 title 'merged datasets';
run;
```

A statistical question is:

The program fragment below illustrates the basic algorithm for creating pseudorandom variables, and thus is the backbone of any simulation. Describe what each of its statements accomplishes. What would go wrong if the "output" statement is omitted? Is this a syntax error or a logical error? Modify the program to create a randomization list for 30 patients, where approximately 70% of the patients are randomized to group A and the remainder are randomized to group B.

```
data rand1;

 call streaminit(1233);

 num_recs=10;

 do id=1 to num_recs;
    random_unif=rand("uniform",0,1);
    output;
 end;

 keep id random_unif;
run;
```

A report-related question is:

Describe how the Output Delivery System (ODS) works. Illustrate how you could capture a printout of the above randomization list as a Word document (which could then be forwarded to an investigator).

An example of an advanced question is:

Consider a character variable of length 10, possibly containing embedded blanks. Create a variable denoting whether this character variable contains any version of "DUKE". It can contain either upper- or lower-case values, and also embedded blanks. For example, ' D uK E999' is acceptable. Make your code as efficient as possible. Even if you are unfamiliar with the details, you may assume that SAS has a reasonably comprehensive set of character functions.

There are numerous ways to solve this problem -- what makes it advanced is that the student must consider questions of efficiency. One efficient solution, described in pseudocode, is: (1) use a character function to translate all the lower-case symbols into upper-case; (2) use a character function to remove the blank spaces; and (3) use a character function to search for 'DUKE'.

Appendix B

A typical examination with commentary (copied from Troy *et al*, 2023)

Question: *What are some similarities and differences between SAS and R?*

Answer: R is an object-oriented functional language organized around lists whereas SAS is a procedurally-based language organized around data frames. R is open-source whereas SAS isn't, an implication of which is that it requires downloading packages that might or might not work as desired. R is more flexible than SAS. Work sessions are different: R code is immediately executed whereas for SAS you write a block of code, mark it, and then execute it. Either language can be used to perform typical data management and statistical analysis tasks.

Comment: Apart from its information value, this question also serves to differentiate between the perspectives of statistical users (e.g., "the two languages can perform similar functions such as data management and analysis") and those with formal computer science training (e.g., "R is an object-oriented functional language whereas SAS is a procedurally-based language").

Question: *What are some techniques for performing literate and reproducible programming in SAS?*

Answer: Modularize the code, include human-friendly practices such as comments, white space and indentation, write code to be generalizable rather than task-specific, have a development and testing plan, use version control, when performing simulations set a seed for the pseudorandom number generator.

Comment: This question addresses general programming knowledge, and is absolutely critical to an actual coding interview. As a rule, we only hire applicants who can describe literate and reproducible programming practices.

Question: *What are some techniques for testing a SAS program?*

Answer: Perform unit testing by checking components of the program one at a time, trace the logic by printing intermediate files and by using PUT statements to print intermediate results of iterative calculations.

Comment: This question also addresses an important element of general programming knowledge, that being the use of an explicit strategy for program development.

Question: *What does the following code do? Your answer should mention three of the four common ways to create a SAS dataset.*

```
data set1;
  input id rec_no y;
datalines;
1 1 5
1 2 6
1 3 4
1 4 8
2 1 0
```

```
2 2 2
2 3 5
2 4 6
;
run;

proc print data=set1;
run;

data set2;
 set set1;
  if (y>5) then ind=1;
  else ind=0;
run;

proc means noprint nway data=set2;
 class id;
 var ind;
 output out=out1
       mean=new_var;
run;

proc print data=out1;
run;
```

Answer: The first DATA statement creates the SAS dataset called SET1 using raw data as input (i.e., method 1). The INPUT statement provides the input directions. The DATALINES statement tells SAS that the raw data follows. The RUN statement defines the end of this logical entity. SET1 should have 8 observations and 3 variables.

The second DATA statement creates the SAS dataset called SET2 (the child) from the SAS dataset SET1 (the parent). It illustrates creating one or more child SAS datasets from one or parent SAS datasets (i.e., method 2). The IF / THEN statements create a new "indicator" variable denoting whether or not Y>5. SET2 should have 8 observations and 4 variables.

The MEANS statement creates a new SAS dataset called OUT1 as the output from a SAS procedure (i.e., method 3). In this case, we define subgroup means, with the ID variable defining the subgroups. The calculations are applied to the variable IND, and the subgroup means are saved to the new dataset in a variable named NEW_VAR. The PRINT statements print the contents of the various SAS datasets. Although I'm not completely certain about this, OUT1 should have one row per subgroup and include variables containing the subgroup name and the subgroup mean.

Comment: This assesses the general programming construct of indicator variables. These are, among others, an essential part of various counting algorithms. It also assesses familiarity with SAS, as inexperienced users are unlikely to have encountered creating SAS datasets as output from a SAS procedure.

Question: *How would you apply the same logic to creating a new SAS dataset containing the predicted values from a linear regression model? You should have encountered this in a first-year data analysis course. An algorithm is sufficient.*

Answer: Use the REG procedure to perform the regression. As part of that procedure, use the MODEL statement to define the predictor and outcome variables. This procedure ought to have an OUTPUT statement like that for PROC

MEANS. Within that OUTPUT statement, there should be similar syntax that changes "MEAN = new variable name" into "PREDICTED VALUES = new variable name". The output dataset should have one row per observation in the original dataset and the variables listed here (among others).

Comment: This can be followed by a web search to find the details about the relevant SAS syntax.

Question: *The following code illustrates the core logic of performing a patient-level simulation. What does it do?*

```
data test1;

 call streaminit(1);

 do i=1 to 10;
  rand_unif=rand("uniform",0,1);
  output;
 end;

run;

proc print data=test1;
run;
```

Answer: The DATA statement creates a SAS dataset called TEST1. The CALL statement sets the seed for the pseudorandom number generator, thus ensuring that the results will be reproducible. The DO loop creates 10 records. The RAND function creates uniform random variables on the interval from 0 to 1. The OUTPUT statement explicitly writes the record. TEST1 will have 10 observations and 2 variables. The PRINT statement prints TEST1.

Comment: The data structure is a 10x2 array. The algorithm uses a DO loop to create simulated patients, and the call to the pseudorandom number generator produces the simulated data according to the desired specifications. Indentation is a literate programming practice. Because this is part of an examination, the literate programming practices of commenting program code and titling output are not illustrated. Setting a seed for the pseudorandom number supports reproducibility.

Question: *What happens if the OUTPUT statement is dropped?*

Answer: The DO loop will still be executed 10 times, but only the final record will be printed.

Comment: This question evaluates general programming knowledge about how DO/FOR loops operate. If a student is stumped, they are encouraged to run the code without the OUTPUT statement and explain the results. A follow-up task could be to report intermediate results using a PUT statement, which would verify that the DO loop is operating as desired but not writing the result.

Question: *How would I simulate a standard normal random variable instead of a uniform random variable?*

Answer: Even though I haven't done so before, the answer ought to be to change "uniform" to "normal" in the RAND function -- the precise syntax might not be precisely as assumed.

Comment: This question evaluates understanding of the basic algorithm that underpins simulations. A correct answer requires understanding that the pseudorandom number is generated by the RAND function, and that the RAND function has options which can be used to specify the desired distribution.

Question: *Suppose that you wanted to prepare a randomization list for a randomized trial, with 40 patients, where patients are randomized to either Intervention or Control with a 1:1 ratio. For any particular list, you won't necessarily have exactly 20 patients in each group. How would you modify the basic simulation code?*

Answer: Change the index on the DO loop from 10 to 40. Use IF / THEN / ELSE logic to assign random numbers which fall between 0 and 0.50 to I and the rest to C. In other words, change the code to this:

```
data test1;

 call streaminit(1);

 do i=1 to 10;
  rand_unif=rand("uniform",0,1);
     if (0 le rand_unif le 0.50) then group='I';
      else group='C';
     output;
 end;

run;

proc print data=test1;
run;
```

Comment: This illustrates a generalizable programming technique, which is essentially independent of language. In algorithmic form: simulate a uniform random variable and then use IF-THEN logic to assign patients to study group based on the value of that uniform random variable. A more efficient solution, specific to this problem, is:

```
data test1;

 call streaminit(1);

 do i=1 to 10;
  group=rand("bernoulli",0.50);
  output;
 end;

run;

proc print data=test1;
run;
```

Question: *An investigator is planning a trial comparing a new smoking cessation intervention with usual care. There will be 40 patients per group. The quit rate in the intervention group is 20% whereas the quit rate in the usual care group will be 10%. What is the estimated statistical power, based on 1,000 simulated replications of the study?*

Answer: From one of the assignments, I know that the general algorithm for using simulation to perform a power calculation is (1) form an outer DO loop covering the 1,000 replications of the study; (2) within an inner DO loop, simulate the data according to the specifications provided by the investigator (here, the result should be a dataset with 80,000 rows (i.e., 1000 iterations time 80 rows per iteration); (3) for each replication, perform a statistical test and output the p-value to a new dataset; (4) map that p-value to a new variable denoting the presence or absence of statistical significance; and (5) the estimated power is the proportion of replications with a statistically significant result. The important specifications are the 20% and 10% quit rates. I'll implement this be cutting and pasting the

general DO loop into one DO loop per study group. The statistical test is a chi-square test, which should be performed by iteration. I need to use some form of OUTPUT statement to write p-values to a new SAS dataset. Instead of looking up the syntax, I just cut and pasted from one of the assignments -- it uses the Mantel-Haentzel version of the chi-square test, which I assume is OK to do here. The resulting code turns out to be:

```
data test1;

 call streaminit(1);

 do iteration=1 to 1000;

  do i=1 to 40;
   group='I';
   quit=rand("bernoulli",0.20);
   output;
  end;

  do i=1 to 40;
   group='C';
   quit=rand("bernoulli",0.10);
   output;
  end;

 end;

run;


proc freq noprint data=test1;
 by iteration;
 tables group*quit / chisq;
 output out=chisq mhchi;
run;


data chisq2;
 set chisq;
 if (p_mhchi<.05) then sig='yes';
 else sig='no ';
run;
```

```
proc freq data=chisq2;
 tables sig;
 title 'task 2';
 title2 'estimated power';
run;
```

Comment: This question requires understanding a general simulation algorithm, which is essentially independent of programming language, then translating it into a data structure that works well in SAS, and then discovering the SAS-specific syntax (e.g., how to OUTPUT p-values from the FREQ procedure). Describing the algorithm demonstrates basic programming skills. If the student is unfamiliar with specific elements of syntax, they are asked to perform a web search and then demonstrate that they can effectively utilize its results.

Question: *How could you make the DO statement more generalizable?*

Answer: Replace the hard-coded value of 40 with a variable such as GROUP_SIZE.

Comment: This illustrates a general programming technique that contributes to literate programming, regardless of language. It also provides a hint for the next question.

Question: *With 40 per group the statistical power is unacceptably low. Suppose that the investigator asks you a slightly different question: namely, how many patients would be needed to achieve 80% power? How would you modify the previous code? An algorithm is sufficient.*

Answer: Replace 40 with GROUP_SIZE, add an outer loop that changes the value of GROUP_SIZE (e.g., DO GROUP_SIZE=40 to 200 by 10;), stop when the desired power is achieved. Indeed, a DO WHILE structure would be more efficient than a standard DO loop.

Comment: This also translates general statistical knowledge into an algorithm, appropriate for SAS.

Acknowledgments

Not applicable.

Authors contributions

GS was responsible for drafting the manuscript. JT performed literature review. MN was the previous instructor of the course described here, and GS utilized various of her teaching materials. As members of a writing group dedicated to research in educational pedagogy, all authors provided critical review and proposed edits. All authors read and approved the final manuscript.

Funding

No external funding was utilized.

Competing interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Informed consent

Obtained.

Ethics approval

The Publication Ethics Committee of the Sciedu Press.

The journal's policies adhere to the Core Practices established by the Committee on Publication Ethics (COPE).

Provenance and peer review

Not commissioned; externally double-blind peer reviewed.

Data availability statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Data sharing statement

No additional data are available.