Using Coding Interviews as an Organizational and Evaluative Framework for a Graduate Course in Programming

Gregory Samsa[1,*]

[1]Department of Biostatistics and Bioinformatics, Duke University, Durham, North Carolina, USA

*Correspondence: 11084 Hock Plaza, Durham NC 27710, USA. Tel: 1-919-613-5212. E-mail: greg.samsa@duke.edu

Received: April 30, 2020 Accepted: June 16, 2020 Online Published: August 20, 2020

doi:10.5430/jct.v9n3p107 URL: https://doi.org/10.5430/jct.v9n3p107

Abstract

Objective: In a Statistical Analysis System (SAS) coding interview, job applicants are typically presented with data management and data analysis problems and asked to solve them using the programming language of SAS. Interviewers not only assess technical competence, but also algorithm design and more generally how applicants approach computer programming. In the language of constructivism, the problems are designed to assess the depth and soundness of the applicant's mental model of SAS programming. We asked whether a SAS course, embedded within a Master of Biostatistics program, could reasonably be structured using a coding interview for the final examination as its organizing framework.

Methods: This is a case study, where we describe how our content delivery was structured in order to prepare students for their coding interviews. It additionally relies on the metaphor of learning a second language through immersion.

Results: Using a constructivist approach enhanced with active learning exercises, a course could in fact be designed around a coding interview. Course content can be mapped to the metaphor of foreign language immersion. Student response has been positive, and the formative evaluation has been encouraging to date.

Conclusions: Coding interviews are a novel and potentially promising way to design a course in SAS programming.

Keywords: biostatistics, coding interview, constructivism, curriculum design, formative evaluation, SA

1. Introduction

At the broadest level, the fundamental question underpinning our program of research is how to most effectively prepare students to practice the discipline of collaborative biostatistics. In the spirit of a cognitive apprenticeship model derived from constructivism, (Bandura, 1997; Johnson, 1992) we began the initial development of a masters' program in biostatistics with a vision-setting exercise whereby we interviewed successful biostatisticians and their collaborators around the question of what constitutes an effective collaborative biostatistician. The answers were consistent across both groups and, indeed, with the modest literature on the topic, and were termed the "ABCs of biostatistics":

• **Analytical skills** include mastery of statistical analysis methods and statistical programming, and also more general skills in problem definition and problem solving.

• **Biology skills** include a combination of substance matter expertise around biology and medicine with curiosity and learning skills to sufficiently master the substance of the content being studied.

• **Communication skills** fundamental to team science include taking in information from outside the discipline and effectively sharing statistical information with others.

In attempting to help students learn the ABCs of biostatistics, our program is organized around a required curriculum in year 1 supplemented by electives in year 2 (and also significant experiential learning opportunities in year 2 through internships and a masters' project). The year 1 curriculum is divided into four academic sequences (plus a professional development sequence): (1) theory; (2) data analysis; (3) biology and statistical practice; and (4) programming. Here, we focus on the second-semester course within the programming track pertaining to the

Statistical Analysis System (SAS) language. (SAS [computer program]. Version 9.4. Cary, NC: SAS Institute Inc; 2014)

The SAS course is preceded by a first-semester course in R, whereby students develop general programming skills which can translate into multiple programming languages, including SAS. At the risk of oversimplification, within biostatistics R is currently the academic standard and SAS is the industry standard. Students who plan to enter industry will eventually need to develop a higher level of proficiency in SAS, and one of the goals of the SAS course is to provide those students with a sufficiently detailed knowledge of the structure of SAS to facilitate their future learning.

The SAS course was included in a recent curriculum review. Student feedback was positive, and a faculty member found the goals and content to be appropriate and the order of its delivery sound. On the other hand, though, we concluded that the course's evaluation methods left something to be desired. We were evaluating students using in-class tests, where we asked them to either write or interpret SAS programs. When the tests were closed-book and closed-technology we worried that we weren't evaluating students in the context that they would actually be using SAS. On the other hand, when the tests were open-book and open-technology we worried that what we were actually evaluating was facility in online searching, not even to mention potential issues around cheating.

Our response was to change the final examination to a simulated "coding interview" – that is, the type of interview often provided to software engineers, statistical programmers, and biostatisticians whose job responsibilities involve a significant amount of programming. In an actual such interview, applicants are typically presented with data management and data analysis problems and asked how they would solve them using SAS. Interviewers not only assess technical competence, but also algorithm design and more generally how applicants approach programming. In the language of constructivism, the problems are designed to assess the depth and soundness of the applicant's mental model of SAS programming. In the language of pragmatism, if nothing else the final examination provides students with practice in an important job-related skill.

We believe coding interviews to be a novel approach for organizing a biostatistics course, and for evaluating student performance. Here, we describe how our content delivery was structured in order to prepare students for their coding interviews. In the process, we utilized a metaphor which, although not strictly necessary, proved quite helpful. The metaphor was teaching a foreign language through immersion. Recognizing that a barrier to coding interviews could be a concern about their potential subjectivity, we also describe how student performance on those interviews was evaluated.

2. Methods

The didactic presentation begins with a description of our conceptual model (i.e., the metaphor of language immersion). Then, the structure of the class is described. Then, as a case study two illustrative modules (i.e., printing, simulation) are presented. Then, we present the results of an initial and formative evaluation of the redesigned SAS course, subdivided according to the coding interview and other elements of the class. Finally, we discuss our tentative conclusions to date.

Our criteria for success considered, among others, the following:

• Can individual classes be designed using the metaphor of language immersion?

• Do students demonstrate good SAS programming skills?

• Can coding interviews be successfully administered as a final examination?

• Do students and instructors evaluate coding interviews positively?

In brief, the answer to all these questions turned out to be "yes".

3. Results

3.1 Conceptual Model for Teaching SAS as a Second Language

Our conceptual model / analytic framework identifies some of the elements of successful instruction in a second language, and then translates them into the context of becoming proficient in SAS. Among these elements are the following:

• Practice speaking the language, beginning with the first lesson

• Use repetition

• Beginning from a simple base, gradually expand the complexity of what is spoken

• Deliver information about language structure in a just-in-time fashion

• Place examples in context

• Use an online dictionary and similar resources

This conceptual model is fundamentally a metaphor which is consistent with the minimalist approach to developing software manuals and similar training materials. (Ben-Ami, 2001) In this context, the fundamental goal is to help the learner develop a sound mental model of how the programming language is organized, and the approach combines immediate immersion to accomplish simple tasks with ongoing explication of the structure of the language. As per constructivism, language structure is initially described in terms which are very simple (albeit sound), with additional detail gradually added over time. At each step, the previous mental model of language structure is used as a building block.

3.1.1 Application to Spanish

3.1.1.1 Practice Speaking the Language, Beginning with the First Lesson

For example, a first lesson in Spanish might begin with the Spanish equivalents of good morning, good evening and good night: *Buenos días, Buenas tardes, Buenas noches*, respectively. The example is chosen to have a consistent sentence structure, and helps to develop vocabulary (e.g., *Buenos/Buenas* is repeated, and becomes one of the first vocabulary words).

3.1.1.2 Use Repetition

As noted above, repetition aids in developing vocabulary and internalizing grammar.

3.1.1.3 Beginning from a Simple Base, Gradually Expand the Complexity of What is Spoken

For example, an element of a first lesson in Spanish could extend *Buenas noches* (good night) to *Buenas noches mi amor* (good night my love).

3.1.1.4 Deliver Information about Language Structure in a Just-in-Time Fashion

Without discussing Spanish grammar in detail, the above example illustrates one way that a Spanish sentence can be structured. It also naturally leads to a just-in-time discussion about the distinction between *buenos* and *buenas*, which illustrates the rule that gender is attached to nouns. This would encourage students to replace the simple conceptualization of "nouns" with "gender-specific nouns", a first step in developing increasingly sophisticated mental models about the structure of Spanish.

3.1.1.5 Place Examples in Context

An early Spanish lesson might focus on an important task, such as introducing one's family. For example, *Esta es mi familia*, *Esta es mi madre*, and *Esta es mi padre,* are this is my family, mother and father, respectively. As above, repetition aids in developing vocabulary and internalizing grammar.

3.1.1.6 Use an Online Dictionary and Similar Resources

Nowadays, learning a foreign language is made significantly simpler by online resources such as dictionaries. For example, not only can individual vocabulary words be searched, but it is straightforward to ask questions such as "how do I say this is my family in Spanish?" The answers to such questions comprise, either directly or indirectly, a significant component of a mental model of the Spanish language.

3.2 Class Structure

3.2.1 Class Sessions

Class sessions are divided between demonstration and practice. Class sessions include:

• Interactive demonstrations of SAS code

• Whiteboarding exercises (e.g., to diagram SAS code, to describe and critique the algorithms underpinning the SAS code).

• Programming assignments.

Demonstrations were substantially based on a previous iteration of the class, and the slide decks from that iteration were included as supplemental resources. Here, the whiteboarding exercises are also intended to illustrate one component of the coding interview. The programming assignments are group-based, and class time is reserved to

begin these assignments with the instructor's assistance. Completing these assignments earns a grade of B for the course, which can be increased to a maximum of an A+ based on the coding interview.

3.2.2 Coding Interview

Students are provided with a grading rubric, an opportunity to practice the coding interview ahead of time, and whiteboarding and similar in-class experiences which also serve to demonstrate and then help students practice designing and explaining SAS programs. The grading rubric was derived from rubrics used for interviews of software designers, (e.g., Haoyi, 2017) and includes questions such as:

• Was the algorithm clearly described?

• Was the algorithm sound?

• Was a development and testing strategy described?

• Were good programming techniques (e.g., modularization) used?

• Was the SAS code documented using comments?

• Was the SAS program user-friendly?

• Was the SAS syntax (for the critical portions of the program) accurate?

• Was the structure of the SAS language (e.g., similarities and differences compared to R) clearly described?

3.3 Case Studies

3.3.1 Description

Here, we illustrate two modules: (1) printing (Appendix 1); and (2) simulation (Appendix 2). Class materials include illustrative SAS code, SAS output, and a narrative summary. Much of the description of the SAS code is contained in the comment boxes (which begin with an asterisk and proceed until a semicolon is encountered). The illustrative code includes references to various in-class exercises.

3.3.2 Printing Module

3.3.2.1 Practice Speaking the Language, Beginning with the First Lesson

Printing is the first module in the course. Rather than beginning the course more traditionally – for example, with an introduction to SAS syntax – we start with the simple task of printing an existing dataset. (The dataset must first be created, but the mechanics around dataset creation are deemphasized.) Accomplishing a simple task such as printing is the functional equivalent of speaking the language, and thus the first module begins with "speaking SAS".

3.3.2.2 Use Repetition

PROC PRINT is repeated in all 11 of the demonstrations in this module, as is the TITLE statement.

3.3.2.3 Beginning from a Simple Base, Gradually Expand the Complexity of What is Spoken

From the most basic PROC PRINT, more complex elements are gradually added – for example, selecting variables to print and selecting observations to print.

3.3.2.4 Deliver Information about Language Structure in a Just-in-Time Fashion

Various elements of the structure of SAS are illustrated within the printing module. For example, a fundamental element is that SAS datasets are either being created or acted upon. Printing is one such action. An element of SAS grammar is that all statements end with a semicolon. Another element of structure is that the user can limit the variables (i.e., columns of the data array) to be acted upon, either through a SAS statement or when referring to a SAS dataset within a SAS procedure.

3.3.2.5 Place Examples in Context

Even though printing is an extraordinarily simple task, it is used to introduce more general programming concepts such as reproducible programming and modular coding, both of which are topics that interviewers might address during a coding interview. Reproducible programming is introduced through the notion of referring to the source program within the printed output. Modular programming (and also an additional element of the SAS language) is introduced through translating a PRINT statement into a macro. An additional general concept is that of reader-friendly code, which is accomplished through liberal use of commenting, white space, indenting, etc.

3.3.2.6 Use an Online Dictionary and Similar Resources

Although the students will have encountered the functional equivalent of a macro in their previous R course, this component of the lesson is notably more sophisticated than others. For macros, the instructor explicitly models the process of learning by first performing a web search on "What does a SAS macro do?", and then using SAS's help function which, quite fortunately, not only includes a structured description of macro syntax but example macros as well. Indeed, once the student has an adequate understanding of the structure of SAS and what a macro does, it is usually sufficient for them to find an example and then modify it to meet their needs. In other words, the student is encouraged to dive in and "speak macro".

3.3.3 Simulation Module

3.3.3.1 Practice Speaking the Language, Beginning with the First Lesson

The most fundamental idea behind a typical simulation is to create a DO LOOP, where each execution of the loop uses a random number generator to create a simulated data value. The module begins with practice creating such a loop (i.e., "speaking" DO LOOP).

3.3.3.2 Use Repetition

The fundamental DO LOOP is repeated for each simulation.

3.3.3.3 Beginning from a Simple Base, Gradually Expand the Complexity of What is Spoken

This module illustrates one of the ways that code can be developed in an iterative fashion which, indeed, is one of the qualities that interviewers are seeking in a typical coding interview. For example, the basic logic of the random number generation is developed and tested on a single iteration of the simulation, and only then expanded to add an outer DO LOOP which repeats that logic on multiple iterations.

3.3.3.4 Deliver Information about Language Structure in a Just-in-Time Fashion

This module illustrates explicit attention to "algorithm structure" – that is, it is organized around the three basic ways to perform a simulation typically encountered within biostatistics. To assist in highlighting the steps, intermediate files are printed (which, indeed, is an excellent programming technique in general). In class, printing intermediate steps is supplemented with a whiteboarding exercise, where the logic of the simulation is first mocked up as a flow chart or pseudocode, and then the actual SAS code is diagrammed.

3.3.3.5 Place Examples in Context

For this module, the context is a task which is critical to the practice of biostatistics: namely, simulation. Indeed, the information pertaining to SAS is supplemented with content about various statistical principles pertinent to designing sound simulations.

3.3.3.6 Use an Online Dictionary and Similar Resources

For this module, students are encouraged to explore web material such as guides to designing simulations, and also SAS's help to obtain information about the different types of random variables that SAS is able to directly simulate. (Essentially any random variable can be indirectly simulated by first defining its cumulative distribution function, and then randomly selecting a value from that function through a uniform random variable in the interval from 0 to 1.).

3.4 Course Evaluation

Best laid plans being what they are, in mid-semester the covid-19 crisis arrived, and in less than a fortnight our course delivery was unexpectedly transformed from in-person to synchronous online. The grading scale was changed to satisfactory / unsatisfactory, and since everyone was up to date with their homework (and thus due at least a B) the final exam became unnecessary for the purpose of assigning grades, although students retained the option to receive letter grades, and those students who preferred to do so participated in simulated coding interviews remotely. In addition, some students who received a satisfactory grade opted to undergo a coding interview for practice, which we interpreted as a vote of confidence regarding its value.

Students also participated in an "exit interview" where they were queried about their level of confidence in their SAS programming skills, discussed self-identified gaps in their SAS knowledge with suggestions for self-study, and also discussed their experiences with the course in general. These interviews provided a significant amount of qualitative information, which, because of the unexpected transition to online instruction, we considered to be part of a formative rather than a summative evaluation.

For the element of the course pertaining to the coding interviews, the main insights from this formative evaluation included:

• The notion of designing a programming course around a coding interview was well received, especially given the importance which many students attach to demonstrable job skills, including interviewing skills.

• Those students who participated in the coding interview found it to be a fair assessment of their programming skills.

• Those students who participated in the coding interview found that the instructor's real-time comments on interview technique (e.g., "when the interviewer asked this question they were attempting to assess x, and so a more effective way to frame your response might have been y) was helpful, and increased confidence in their ability to successfully navigate an actual interview.

• Particularly effective programming interviews should be recorded and used as study guides going forward.

For the other elements of the course, the main insights from this formative evaluation included:

• Active practice is critical for mastering programming – even if the techniques in question are straightforward simply seeing them isn't enough.

• An ideal assignment first allows students to practice programming techniques covered during the class as a check on understanding, then requires a modest extension of those techniques, and also embeds some type of practice in explaining the algorithms in question.

• The collaborative component of the homework exercises is enjoyable and helpful – if the course is delivered online, techniques such as instructor-moderated chat rooms should be used to better approximate the experience of working together in real time.

• A substantial project, whereby students perform data management and then data analysis on a dataset which is larger than those used for "toy" examples, would help to assess SAS programming skills within a realistic context.

• A diversity of background materials is beneficial. Some students found the PowerPoint slides from the previous iteration of the class to be useful, and suggested that this be used as a supplemental mode of instruction during class time. Others found the narrative summaries to be helpful. Finding model code (whether from SAS's documentation or the internet) and then modifying it is common practice, and exercises which allow students to systematically practice this skill should be considered.

4. Discussion

We have described a SAS programming class with a novel approach to evaluating the performance of its students: namely, a coding interview. The formative evaluation of this approach has been encouraging to date.

When discussing this approach with colleagues, three quite reasonable questions have been raised about using an interview for a final examination: (1) is it fair; (2) will students object; and (3) is it too time consuming for the instructor? In our experience using interviews (albeit not coding interviews) in other courses, the first two questions are related, as in: will students believe that an interview is subjective, potentially unfair, and thus object?

Our previous experience with interviews has shown that, although a minority of students find the experience to be frightening to contemplate ahead of time, no one has ever objected to their final grade. Most likely this is because, with the opportunity to ask follow-up questions and otherwise probe, the limits of a student's understanding are apparent not only to the instructor, but to the student as well.

We also attempt to be as transparent as possible about the coding interview. For example, we circulate a grading rubric, sample questions, and differentiate between which elements of SAS are core knowledge, and thus should be recalled during an interview, and which are not. Using the simulation module for example, the notion of a DO LOOP with an OUTPUT statement to write the value of the simulated random variable can be considered to be core knowledge, whereas the name of the RANNOR function (which generates values of a normal random variable) and its syntax is not. Thus, a perfectly acceptable response within a coding interview would be to use actual SAS code to set up the DO LOOP, and pseudocode to call the function (e.g., as in "apply SAS's function which generates normal random variables – I don't recall its syntax but can look it up if you like – and write the results to a variable named Y"). After all, on the job the students can always run a web search such as "SAS functions, normal random variable".

Regarding the third question, a less than collegial response might be: "Don't students pay a lot of tuition and aren't we quite well paid to teach?" A more effective response, however, is: "Actually, with creative scheduling the

logistics aren't so bad. The practice interviews don't have to wait until reading week, and instead can often be scheduled any time during the last quarter of the semester. The students who ace the practice interviews (who tend to be the ones who sign up early) can use those interviews in lieu of the final, and so the volume of interviews during the final examination period probably won't be all that burdensome. Not even to mention that time passes quickly as the students are lots of fun to talk with."

One point of emphasis within our overall curriculum is training students to "think like biostatisticians" which, in the constructivist/ cognitive apprenticeship paradigm, is functionally equivalent to "develop a mental model of the discipline that is sound, and roughly approximates the mental models used by experienced biostatisticians". Achieving this goal in general is not a trivial matter, nor is determining how this construct can be best evaluated.

We would argue that a programming course is an especially felicitous use case for a constructivist approach, as it "involves the production of an artifact that can be shown and shared" (Monga *et al*, 2018). A SAS program either works as intended or it doesn't, and thus provides immediate feedback to the student about the soundness of their thinking. Moreover, the underlying logic of a SAS program that a student has written can be translated into an algorithm which can be depicted on a whiteboard, and that translation is only one step removed from the student's mental model of the SAS language. Indeed, it is a direct application of that mental model, and is sufficiently concrete as to provide actionable information about the depth and soundness of that mental model.

The didactic elements of the course are intended to be a methodical "show and tell" illustrating how to design and code SAS programs. In creating those elements, we found the metaphor of language immersion to be both novel and helpful, although not necessarily crucial, and in the final analysis this metaphor is only helpful to the extent that it assists in creating sound course materials. We found it beneficial to build short exercises into the didactic presentation, as these encourage active engagement, and also help students to assess when their understanding of a construct is as desired. The collaborative homework assignments are intended as another application of the principles of active learning.

As previously noted, this course unexpectedly received a stress test, in the form of being ported to synchronous online instruction on very short notice. Assessing the results of this stress test, the didactic elements translated quite well. Indeed, the only major change was that more extensive narrative summaries were created and circulated ahead of time, and directly linked to the SAS program via numbered "demonstrations". Even if the course format returns to an in-person one, we will retain detailed narrative summaries. The hands-on exercises didn't port as well, as it was difficult to recreate the experience of small groups of students working on a SAS program assigned as homework with the instructor circulating to offer guidance. In retrospect, this is a problem that can mostly be addressed using available technology, and our struggles are attributable to a learning curve in the presence of multiple distractions. The coding interview translated fairly well, the main challenge being the lack of a functional equivalent of a whiteboard. This is also a problem which can be addressed using available technology, and ensuring that all our students have the ability to write on their computer screens and share the results with others would be a top priority if the SAS course is moved online in the future.

In summary, we find this approach to software instruction to be novel and encouraging, and look forward to making improvements and then performing a more formal evaluation in the future.

5. Conclusion

It can be asked what have we learned, and why does it matter? Briefly, what we learned is that designing a SAS course around a coding interview "works", at least for this particular audience. However, given the proliferation of materials for learning SAS (e.g., online courses, in-person short courses, certification materials designed by SAS personnel), it can certainly be asked why this course might be "better" than other options, or whether an additional approach to teaching SAS, even if novel, is needed at all. Indeed, in the extreme one might even ask whether separate courses in programming are necessary within a biostatistics curriculum – perhaps the time might be better spent teaching statistical techniques with programming encountered as needed?

We would argue that our approach is only "better" to the extent that it is intentionally targeted toward its specific audience. Our students aren't training to become professional programmers, nor will they use all of SAS's functionality. Instead, they are training to become biostatisticians operating within an environment of team science. This requires "tourist-level" familiarity with SAS to allow them to perform standard data management and data analysis tasks in a clear, reproducible, and human-friendly fashion (e.g., since others will encounter their SAS code). They also need to be able to accomplish various tasks which are high on the level of statistical sophistication and

moderate on the level of programming sophistication – for example, performing non-standard sample size calculations, comparing the statistical implications of various study designs, comparing various statistical techniques, etc.

One element of this targeting is to deliver content in a fashion which treats our students as biostatisticians rather than professional SAS programmers. Another element is to focus on different topics that a typical SAS course outside biostatistics covers, the simulation module providing an example. Thus, we can promise our students that "these are the elements of SAS which you are most likely to use in practice, and this is the context within which you are likely to use them". Actual interviewers are trying to assess much the same construct – namely, whether student's mental model of SAS is sound and sufficient for the task at hand. "What the student needs to be able to do" equates to "what the interviewer will ask", and this allows the instructor to get the emphasis right and similarly helps motivate students to learn. Interestingly, the interviewer becomes a personification of the teaching goals.

To appreciate why this "matters", it can be helpful to consider the value proposition for biostatisticians. Similar to so many knowledge-based professions, the value proposition for biostatisticians is that they "think like biostatisticians". Factual knowledge, such as how to perform a particular mathematical derivation, is now ubiquitous (or at least potentially so) through the internet, and biostatisticians only add value via their ability to more insightfully process information which is also known to others. Given this value proposition, it seemed only natural to design our curriculum to address the construct of a sound mental model of biostatistics as directly as possible, and also to apply this philosophy to redesigning the SAS course.

When performing due diligence around the question of "how do I teach someone to think like a biostatistician", a constructivist notion which is also consistent with the above value proposition, we discovered that the literature on this question was sparse to the point of being non-existent. Certainly, the literature on constructivism is voluminous, and includes insights about how to teach statistics and how to teach computer programming (although not necessarily for our particular audience). (e.g., Ben-Ami, 2001; Wulf, 2005) Within the statistical literature, there is a modestly sized literature on how to assist students to develop skills in consultation and team science – although this isn't quite "how to think like a biostatistician" it touches on closely related constructs and is quite helpful (e.g., Samsa, 2018 provides a reference list). There is a massive literature on how to teach computer programming, although not necessarily for our specific audience. (Monga *et al*, 2018) provides a helpful reference list oriented toward introductory programming classes, although it must be acknowledged that our pedagogic task is somewhat different in that we are trying to simultaneously trying to teach statistics and programming.

Thus, we would argue that the fundamental question underpinning our program of research about how to most effectively prepare students to practice the discipline of collaborative biostatistics is both important and understudied. We would also argue that operationalizing this question as how to most effectively prepare students to think like biostatisticians represents progress. We would further argue that a course in SAS programming offers an excellent test case, because the student's mental model (or, more accurately, something only one step removed from that mental model) is directly observable (e.g., as an algorithm, as a SAS program). The general pedagogic insight which we would offer to others is that, where possible, a coding interview or its functional equivalent can be effective for purposes of both curriculum design and student evaluation, and should be considered. Certainly, such interviews are consistent with the notion that it is more important to test the ability for students to use information than to recall facts. We would argue that this illustrates the general principle that it is beneficial to synchronize curriculum design and student evaluation as much as is realistically feasible. Finally, we would offer encouragement to other instructors that coding interviews and their functional equivalents are realistic and, indeed, can be enjoyable.

For readers intending to develop a similar class, we recommend Monga *et al*, 2018 as general background about a constructivist approach to teaching programming, (Alammary, 2019) as a useful systematic review of the use of blended learning models for introductory programming courses, (Eason, 2004) as an interviewer-based perspective on the design of SAS coding interviews, and a web search on "key SAS interview questions you should know", as providing both interviewing tips for students and sample questions for the coding interview. These latter two resources can be quite helpful to share with students.

References

Alamarry, A. (2019). Blended learning models for introductory programming courses: A systematic review. *PLoS ONE, 14*(9), 30221765. https://doi.org/0.1371/journal.pone.0221765

Bandura, A. (1997). *Social Learning Theory*. Englewood Cliffs, NJ: Prentice-Hall.

Ben-Ami, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45-73.

Eason, J. (2004). *Assessing SAS skill level during the interview process*. SAS Users Group International Conference, May 9-12, 2004. Montreal, Canada. Paper 133-29.

Haoyi (2017). How to conduct a good programming interview. Retrieved 15 June, 2020 from https://www.lihaoyi.com/post/HowtoconductagoodProgrammingInterview.html

Johnson, S. D. (1992). A framework for technology education curricula which emphasizes intellectual processes. *Journal of Technology Education, 3*, 1-11. https://doi.org/10.21061/jte.v3i2.a.4

Monga, M., Lodi M., Malchiodi D., Morpurgo A., & Spieler B. (2018). *Learning to program in a constructionist way*. Proceedings of Constructionism 2018, Aug 2018, Vilnius, Lithuania. hal-01913065.

Samsa, G., LeBlanc, T. W., Locke, S. C., & Pomann, G. M. (2018). 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 [computer program]. Version 9.4. Cary, NC: SAS Institute Inc; 2014.

Wulf, T. (2005). Constructivist approaches for teaching computer programming. *SIGITE'05,* 245-249, October 20-22, 2005. Newark, New Jersey, USA. https://doi.org/10.1145/1095714.1095771

Appendix 1a: Printing Module Narrative Summary

Demonstration 1 prints the SAS dataset WORK.TEMP1. It illustrates the most basic syntax for printing a dataset. Any execution of PROC PRINT should include an explicit pointer to the dataset being printed (here, via the DATA=option). The output from all SAS procedures should be titled, via a TITLE statement.

Demonstration 2 illustrates multiple title statements, and how you can title your output to include a reference to the program which created it.

Demonstration 3 illustrates that you don't have to print all the variables within the SAS dataset, and various syntax options for naming the variables that you want to print within a VAR statement.

Demonstration 4 illustrates that you can control the variables to be processed through the DROP and KEEP dataset options. In other words, these options let you select variables from a dataset on the fly.

Demonstration 5 illustrates that variables need not be printed using their default format, and that this format can be controlled by the programmer. A more detailed treatment of this topic is contained in the module on formatting.

Demonstration 6 illustrates how to add a summation to selected variables within the printout using the SUM statement.

Demonstration 7 illustrates one way to filter observations (i.e., print only the subset of a dataset whose values meet a certain condition). A WHERE statement is used to perform the filtering. The logic of filtering, including statements such as IF, THEN, ELSE and WHERE is essentially identical to what you have already encountered in R (although the syntax is slightly different).

Demonstration 8 illustrates how to filter on the observation number. One approach is to use FIRSTOBS and OBS when referring to the dataset to be printed. Another approach is to use the SAS variable _N_.

Demonstration 9 illustrates one way to retain the observation number as part of a newly created SAS dataset. This programming idea will be used in other contexts as well. We also illustrate the use of the NOOBS option on the PRINT statement to suppress printing the observation number.

Demonstration 10 illustrates printing the dataset in groups, using a BY statement. The dataset must first be sorted, using PROC SORT, also with a BY statement.

Demonstration 11 illustrates how a macro can be used to automate the printing process. This is only an introduction to the idea of a macro, which has an analog in R. Macros are covered in more detail in a separate module.

Appendix 1b: Printing Module SAS Code

```
************************************************************************    *
* Module 1: printing                                                    *
************************************************************************    *;
options ls=80;
run;
************************************************************************    *
* One way to add comments to a SAS program is to start with an asterisk.    *
* Anything between the asterisk and the next semicolon is considered to be a    *
* comment.    In the program window, comments are colored green.            *
* Making the comments into a box is one way to highlight them.              *
************************************************************************    *;

************************************************************************    *
* This module focuses on printing SAS datasets.    In SAS, you are either creating a    *
* SAS dataset (e.g., through a DATA statement) or analyzing a SAS dataset   *
* (i.e. using a PROC statement.)    Programming statements aren't case-sensitive.    *
* As a convention, I will capitalize the DATA and PROC statements in order to    *
* highlight their fundamental nature. PROC is short for procedure.          *
************************************************************************    *;

************************************************************************    *
* A simple DATA statement creates a temporary SAS dataset called work.temp1.    *
* It is located in the work folder (which has a default location, and can be moved if    *
* desired -- I won't demonstrate how right now).    The work folder only lasts for the    *
* duration of the SAS session.                                              *
************************************************************************    *;

************************************************************************    *
* The structure of this DATA statement is as follows.                       *
* DATA names the SAS dataset being created to be work.temp1.                *
* INPUT says that the variables being created are named x, y, and z         *
* (in the absence of an explicit indicator for a character variable         *
* x, y, and z are considered to be numeric).                                *
* DATALINES says the data values follow within the program.                 *
* There are many other ways to create SAS datasets (to be discussed later).    *
* For now, all we want to do is to create one in a straightforward manner.  *
* The semicolon at the end of the data marks the end of the input task.    (If the    *
* data contain semicolons there is a work-around (not illustrated here.)    *
* The RUN statement tells SAS to execute the DATA statement.                *
************************************************************************    *;
```

```
**********************************************************************  *
* All SAS datasets have a folder location, which is the first-level name (in this case:  *
* work), and a file name within that folder (in this case: temp1).  *
The syntax is foldername.datasetname.  *
* For temporary files in the work directory most programmers omit the first-level  *
* name (i.e., by referring to the file as temp1 rather than work.temp1).  I will keep  *
* the first-level name, in order to emphasize that all SAS datasets in fact have two  *
* parts to their name.  *
**********************************************************************  *;

**********************************************************************  *
* The interface allows you to view work.temp1 as if it were a spreadsheet.  *
**********************************************************************  *;

DATA work.temp1;
input x y z;
datalines;
1 5 1
2 7 1
3 8 1
4 10 1
5 15 0
6 16 0
7 18 0
8 21 0
9 22 0
10 23 0
run;
**********************************************************************  *
* It's good programming practice to review the log after each step, even if you're  *
* confident that the program will work as desired. Here, the log says that work.temp1  *
* has 10 observations and 3 variables, as intended.  *
**********************************************************************  *;

**********************************************************************  *
* We will begin by using the PRINT procedure to print the SAS dataset.  The first  *
* version of the program illustrates the results of running the PRINT procedure  *
* without any options (other than a title).  Technically, I have included a  *
* data= option, in order to explicitly say that the dataset to be printed is work.temp1.  *
* By default, SAS prints the most recently created dataset.  Since only one data set  *
* has been created, PROC PRINT would also work.  This, however, is poor  *
```

```
* programming practice -- you should always explicitly name the SAS dataset being    *
* acted upon.    Otherwise, you might accidentally act upon the wrong dataset.       *
*************************************************************************    *;

*************************************************************************    *
* Exercise: Describe the output.    What information has SAS added, above and    *
* beyond the actual data values?                                                  *
*************************************************************************    *;

*************************************************************************    *
* It is good practice to title all output using a TITLE statement.              *
*************************************************************************    *;
PROC PRINT data=work.temp1;
   title 'Demonstration 1: printout of temp1';
run;
*************************************************************************    *
* Multiple title statements are possible.    Indeed, a simple archiving system would *
* point to the location of the program used to generate the output from the PRINT *
* procedure.    That way, someone holding the printed output could always find the *
* program used to generate it.    There are more sophisticated ways to accomplish *
* this task (not discussed here).    Linking output with the program used to generate *
* that output is one of the key ideas behind reproducible programming.          *
*************************************************************************    *;

*************************************************************************    *
* Another good programming practice is to put the program name at the start of the *
* program, within a comment block.    This allows someone with a printout of the *
* program (or the log) to find it and rerun if necessary.    The SAS program can be *
* saved using the FILE tab within the PROGRAM window.                           *
*************************************************************************    *;
PROC PRINT data=work.temp1;
   title 'Demonstration 2';
   title2 'Printout of temp1';
   title3 'This was generated by the program "module 1 printing", located in directory <named>';
run;
*************************************************************************    *
* Rather than printing the entire dataset we can print pieces.    The pieces can be a *
* subset of the variables (columns), the observations (rows), or both.    This *
* illustrates how to print a subset of the variables, through the VAR (short for *
* variables) statement.                                                          *
*************************************************************************    *;
```

```
PROC PRINT data=work.temp1;
   var x y;
   title 'Demonstration 3';
   title2 'Illustrate explicit selection of X and Y';
run;
*************************************************************************    *
* SAS has various options for listing variables (i.e., so that you don't have to name    *
* every variable within a large list).    This illustrates one way to do so.    *
*************************************************************************    *;

*************************************************************************    *
* Exercise: when could the syntax x--z lead to unexpected results?    *
* How could problems be avoided?    *
*************************************************************************    *;
PROC PRINT data=work.temp1;
   var x--z;
   title 'Demonstration 3';
   title2 'Illustrate explicit selection of X through Z';
run;
*************************************************************************    *
* It is sometimes helpful to refer to all variables, all character variables, all numeric    *
* variables, etc.    This illustrates the use of the _all_ convention to refer to all    *
* variables in the dataset.    *
*************************************************************************    *;
PROC PRINT data=work.temp1;
   var _all_;
   title 'Demonstration 3';
   title2 'Illustrate selection of all variables';
run;
*************************************************************************    *
* This is a slightly more efficient way to make the selection.    Here, the subset    *
* operation is performed within the dataset rather than within the procedure.    This    *
* sort of nuance only tends to matter for very large datasets.    We first use a keep    *
* statement, then a drop statement.    *
*************************************************************************    *;
PROC PRINT data=work.temp1 (keep=x y);
   title 'Demonstration 4';
   title2 'Illustrate explicit selection of X and Y with a KEEP data set option';
run;
PROC PRINT data=work.temp1 (drop=z);
   title 'Demonstration 4';
```

```
   title2 'Illustrate explicit selection of all variables except Z with a DROP data set option';
run;
*************************************************************************     *
* Printed variables can be formatted, which is illustrated in detail elsewhere.     *
* X is printed with 1 and then 2 decimal points.                                     *
*************************************************************************     *;
PROC PRINT data=work.temp1;
   var x--z;
   format x 4.1;
   title 'Demonstration 5';
   title2 'Illustrate formatting, X has 1 decimal point';
run;
*************************************************************************     *
* Note: Illustrate what goes wrong with 3.1, 2.1 and 1.1 formats.                    *
*************************************************************************     *;
PROC PRINT data=work.temp1;
   var x--z;
   format x 5.2;
   title 'Demonstration 5';
   title2 'Illustrate formatting, X has 2 decimal points';
run;
*************************************************************************     *
* At times, it can be helpful to add a sum of selected variables to the printout.    This     *
* illustrates using the SUM statement to do so.                                      *
*************************************************************************     *;
PROC PRINT data=work.temp1;
   var x y z;
   sum x z;
   title 'Demonstration 6';
   title2 'Illustrate summation for X and Z';
run;
*************************************************************************     *
* We now illustrate printing a subset of the observations.                           *
* The WHERE function is used to perform the filtering on the fly. The logical        *
* condition within the WHERE statement could potentially be rather complex.          *
* For clarity, the logical expression is enclosed within parentheses.                *
* More generally, an important element of reproducible programming is writing        *
* programs in a way that's easy for humans to follow – that is, unless the program is     *
* very resource-intensive, efficiency of communication is more important than        *
* execution.    The use of comments, indenting, and blank lines are examples of       *
* techniques that support efficiency of communication.                               *
```

```
*************************************************************************     *;
PROC PRINT data=work.temp1;
   where (x>5);
   var x y;
   sum x;
   title 'Demonstration 7';
   title2 'Illustrate filtering, X>5';
run;
*************************************************************************     *
* Exercise: Why are the obs 6-10 rather than 1-5?                             *
* What rule is SAS applying?                                                  *
*************************************************************************     *;
PROC PRINT data=work.temp1;
   where (x>5 and y>20);
   var x y;
   sum x;
   title 'Demonstration 7';
   title2 'Illustrate filtering, X>5 & Y>20';
run;
*************************************************************************     *
* We can also filter based on the observation number within the dataset.    This code     *
* selects rows 3 through 7, which is fine if what you always want to do is to select     *
* those particular rows. But, otherwise the code might not perform as intended.          *
*************************************************************************     *;

*************************************************************************     *
* As a principle of reproducible programming, programs should be as general as          *
* possible.    In the previous example, suppose that the records with x>5 and y>20       *
* happened to be located in records 3-7. If the goal is to print records with x>5 and   *
* y>20, selecting records 3-7 would work on this particular dataset, but probably not   *
* on another.                                                                           *
*************************************************************************     *;

*************************************************************************     *
* You might want to print a few records when you are restructuring a dataset and        *
* require multiple steps in order to do so.    Then, printing a few records from each   *
* intermediate dataset helps you to visualize the restructuring process, and to verify  *
* that it is proceeding as intended.                                                    *
*************************************************************************     *;
PROC PRINT data=work.temp1 (firstobs=3 obs=7);
   var x y;
```

```
   title 'Demonstration 8';
   title2 'Records 3 through 7';
run;
*************************************************************************     *
* The same idea can be executed using one of SASs automatic variables, albeit at the     *
* cost of having to create another dataset.                                  *
* Here, _n_ exists for the duration of the DATA step only.                   *
*************************************************************************     *;
DATA work.temp2;
  SET work.temp1;
  if (3 le _n_ le 7);
run;
*************************************************************************     *
* Exercise: Why has the observation number changed on the printout?          *
*************************************************************************     *;
PROC PRINT data=work.temp2;
   var x y;
   title 'Demonstration 8';
   title2 'Illustrate filtering, records 3-7';
run;
*************************************************************************     *
* If you want to retain the value of _n_ an assignment statement is required.     *
* Assignment statements read right to left.    Here, OBS is assigned the same value     *
* as _n_. It can sometimes be helpful to keep track of the original row number of a     *
* record as data management proceeds. Line 4 uses an IF statement to perform the     *
* filtering.                                                                 *
*************************************************************************     *;
DATA work.temp2;
  SET work.temp1;
  obs=_n_;
  if (3 le _n_ le 7);
run;
PROC PRINT data=work.temp2;
   var obs x y;
   title 'Demonstration 9';
   title2 'Illustrate filtering, records 3-7, with record number retained';
run;
*************************************************************************     *
* Same printout, but with the line number dropped from the printout.         *
* The NOOBS option accomplishes this.                                        *
* The HELP menu describes all the possible options for the PRINT statement.  *
```

```
* The examples are especially helpful.                                    *
************************************************************************  *;
PROC PRINT data=work.temp2 noobs;
   var obs x y;
   title 'Demonstration 9';
   title2 'Illustrate filtering, records 3-7, with record number retained';
run;
************************************************************************  *
* The dataset can be printed separately by categories of z (i.e., using the BY  *
* statement).  You can sort by multiple variables at once.  It is good programming  *
* practice to use different dataset names for the inputs and outputs of the SORT.  *
* Otherwise, if something goes wrong with the sort you are at risk for losing your  *
* original dataset.  The SORT procedure performs the sorting, the input dataset is  *
* in the DATA= option and the output dataset is named in the OUT= option.  *
************************************************************************  *;
PROC SORT data=work.temp1 out=work.sort1;
   by z;
run;
PROC PRINT data=work.sort1;
   by z;
   var x y z;
   sum x;
   title 'Demonstration 10';
   title2 'Illustrate use of the BY statement';
run;
************************************************************************  *
* A core notion in programming is that if you find yourself doing the same thing  *
* repeatedly then there probably is a better way to proceed.  Using macros might  *
* serve to make the programming more efficient or perhaps easier to follow.  This  *
* is an example of a very simple macro for printing. Use of macros can contribute to  *
* modular programming.  This excellent programming practice is discussed later,  *
* but for now we note that programs are easier to understand (and also debug and  *
* develop) f they are broken into small self-contained pieces.  *
************************************************************************  *;

************************************************************************  *
* The %MACRO statement defines a macro named PRNT.  This macro  *
* effectively passes 3 parameters: dsn, var and title.  Within the macro, the  *
* parameter dsn is denoted by &dsn, and so forth.  *
* The %MEND statement denotes the end of the macro.  *
************************************************************************  *;
```

```
************************************************************************     *
* Exercise: Diagram how the MACRO works.                                     *
* Which variables are local and which are global?                            *
************************************************************************;
%MACRO prnt(dsn,var,title,title2);
   PROC PRINT data=&dsn;
      var &var;
      title &title;
      title2 &title2;
   run;
%MEND;
************************************************************************     *
* The first time that the macro is invoked, the macro variable dsn is assigned the     *
* value work.temp1 (i.e., because dsn is the first parameter to be passed), etc.       *
************************************************************************     *;

%prnt(work.temp1,x,Demonstration 11,Printout);
```

Appendix 1c: Printing module SAS output

Demonstration 1: Printout of temp1

Obs	x	y	z
1	1	5	1
2	2	7	1
3	3	8	1
4	4	10	1
5	5	15	0
6	6	16	0
7	7	18	0
8	8	21	0
9	9	22	0
10	10	23	0

Demonstration 2

Printout of temp1

This was generated by the program "module 1 printing", located in directory <named>

Obs	x	y	z
1	1	5	1
2	2	7	1
3	3	8	1
4	4	10	1
5	5	15	0
6	6	16	0
7	7	18	0
8	8	21	0
9	9	22	0
10	10	23	0

Demonstration 3

Illustrate explicit selection of X and Y

Obs	x	y
1	1	5
2	2	7
3	3	8
4	4	10
5	5	15
6	6	16
7	7	18
8	8	21
9	9	22
10	10	23

Demonstration 3

Illustrate explicit selection of X through Z

Obs	x	y	z
1	1	5	1
2	2	7	1
3	3	8	1
4	4	10	1
5	5	15	0

Obs	x	y	z
6	6	16	0
7	7	18	0
8	8	21	0
9	9	22	0
10	10	23	0

Demonstration 3

Illustrate explicit selection of all variables

Obs	x	y	z
1	1	5	1
2	2	7	1
3	3	8	1
4	4	10	1
5	5	15	0
6	6	16	0
7	7	18	0
8	8	21	0
9	9	22	0
10	10	23	0

Demonstration 4

Illustrate explicit selection of X and Y with a KEEP data set option

Obs	x	y
1	1	5
2	2	7
3	3	8
4	4	10
5	5	15
6	6	16
7	7	18
8	8	21
9	9	22
10	10	23

Demonstration 4

Illustrate explicit selection of all variables except Z with a DROP data set option

Obs	x	y
1	1	5
2	2	7
3	3	8
4	4	10
5	5	15
6	6	16
7	7	18
8	8	21
9	9	22
10	10	23

Demonstration 5

Illustrate formatting, X has 1 decimal point

Obs	x	y	z
1	1.0	5	1
2	2.0	7	1
3	3.0	8	1
4	4.0	10	1
5	5.0	15	0
6	6.0	16	0
7	7.0	18	0
8	8.0	21	0
9	9.0	22	0
10	10.0	23	0

Demonstration 5

Illustrate formatting, X has 2 decimal points

Obs	x	y	z
1	1.00	5	1
2	2.00	7	1
3	3.00	8	1
4	4.00	10	1
5	5.00	15	0
6	6.00	16	0

Obs	x	y	z
7	7.00	18	0
8	8.00	21	0
9	9.00	22	0
10	10.00	23	0

Demonstration 6

Illustrate summation for X and Z

Obs	x	y	z
1	1	5	1
2	2	7	1
3	3	8	1
4	4	10	1
5	5	15	0
6	6	16	0
7	7	18	0
8	8	21	0
9	9	22	0
10	10	23	0
	55		**4**

Demonstration 7

Illustrate filtering, X>5

Obs	x	y
6	6	16
7	7	18
8	8	21
9	9	22
10	10	23
	40	

Demonstration 7

Illustrate filtering, X>5 & Y>20

Obs	x	y
8	8	21
9	9	22
10	10	23

27

Demonstration 8

Records 3 through 7

Obs	x	y
3	3	8
4	4	10
5	5	15
6	6	16
7	7	18

Demonstration 8

Illustrate filtering, records 3-7

Obs	x	y
1	3	8
2	4	10
3	5	15
4	6	16
5	7	18

Demonstration 9

Illustrate filtering, records 3-7, with record number retained

Obs	obs	x	y
1	3	3	8
2	4	4	10
3	5	5	15
4	6	6	16
5	7	7	18

Demonstration 9

Illustrate filtering, records 3-7, with record number retained

obs	x	y
3	3	8
4	4	10
5	5	15
6	6	16
7	7	18

Demonstration 10

Illustrate use of the BY statement

z=0

Obs	x	y	z
1	5	15	0
2	6	16	0
3	7	18	0
4	8	21	0
5	9	22	0
6	10	23	0

z 45

z=1

Obs	x	y	z
7	1	5	1
8	2	7	1
9	3	8	1
10	4	10	1

z 10

55

Demonstration 11

Printout

Obs	x
1	1
2	2
3	3
4	4
5	5
6	6

Obs	x
7	7
8	8
9	9
10	10

Appendix 2a: Simulation Module Narrative Summary

Demonstration 1 illustrates two fundamental elements of a simulation. The first is the use of a SAS function to create a random variable (here, with a normal distribution). The second is the use of a DO LOOP and an OUTPUT statement to write the desired number of copies of the random variable to a new dataset.

Demonstration 2 illustrates using PROC MEANS to not only calculate a maximum value, but to capture it for subsequent use through an OUTPUT statement. Of course, forgetting the output statement causes the DO LOOP to run but not write the records as desired (except the last one).

Demonstration 3 illustrates the most direct way to perform the simulation. An outer DO LOOP has been added, which allows the programmer to program the number of iterations of the simulation. The PROC PRINT only prints enough records to verify that the DO LOOPs are working as desired. A BY statement has been added to the PROC MEANS, in order to calculate a maximum value for each iteration. Remember to check the SAS log to verify that the datasets have the intended number of records. The NOPRINT option on PROC MEANS suppresses the printing, which is helpful if the number of iterations is large and thus that element of the printed output extensive. An advantage of this approach is that it is straightforward. A disadvantage of this approach is that the number of records can proliferate.

Demonstration 4 illustrates the most direct implementation of another approach, one with one record per iteration. Each simulated variable is given a different name, which is facilitated using an ARRAY statement. An advantage of this approach is that a SAS function (here, MAX) can be efficiently used to make the calculations within each iteration. A disadvantage of this approach is that the number of variables can proliferate.

Demonstration 5 illustrates an approach that works for large number of iterations and large number of variables. Calculations are made on the fly, and the method is only appropriate when the quantity to be estimated can be calculated from the current record and a summary of previous records (for example, the current record and the maximum of the previous records). This code also illustrates a more general programming technique, whereby bins are created, initialized to zero, and incremented whenever a condition holds true (i.e., “counting bins”)

Demonstration 6 illustrates how to create a user-defined function. These can be helpful for modularizing code.

Appendix 2b: Simulation Module SAS Code

```
**********************************************************************  *
* Module 5: simulations.                                               *
* This module builds upon the previous module introducing functions.   *
* We illustrate using random number functions to build three types of simulations.  *
**********************************************************************  *;
options ls=80;
run;
**********************************************************************  *
* SAS has a large number of mathematical functions.                    *
* The program simulates 10 normal variables, each with mean 50 and standard deviation 10,  *
* and then calculates the maximum. The RANNOR function generates a random value  *
* (from a standard normal variable).  The argument is the seed, and so if you repeat the seed  *
```

```
* the results should be identical. The MEANS procedure is used to capture the maximum      *
* value and write it to a SAS dataset.  It's a good idea to limit the number of observations  *
* being printed, as simulated datasets can grow quite large.  It's also a good idea to develop  *
* your simulation on a small dataset and then, once it is tested, to increase the sample size.  *
* Here, the development is performed on a dataset with only 10 records.                    *
*************************************************************************          *;
DATA work.simulate;
   do i=1 to 10;
      random=(rannor(1)*10) + 50;
      output;
   end;
run;
PROC PRINT data=work.simulate;
   var i random;
   format random 5.2;
   title 'demonstration 1';
   title2 'Use of a DO loop and a random variable function to write 10 records';
run;
PROC MEANS data=work.simulate max mean std maxdec=2;
   var random;
   output out=work.max
            max=max;
   title 'Demonstration 2';
   title2 'Calculate and output the maximum';
run;
PROC PRINT data=work.max;
   format max 5.2;
   title 'Demonstration 2';
   title2 'Verify that the maximum written has been written to a new dataset';
run;
*************************************************************************          *;
* Exercise: What goes wrong if we forget the output statement?                        *
* Diagramming the code can help clarify the problem.                                  *
*************************************************************************          *;
DATA work.simulate;
   do i=1 to 10;
      random=(rannor(1)*10) + 50;
   end;
   run;
PROC PRINT data=work.simulate (firstobs=1 obs=10);
   format random 5.2;
```

```
   title 'Demonstration 2';
   title2 'Simulated dataset, forgetting the output statement';
run;
*********************************************************************          *
* Generate 30 iterations of the simulation using nested DO LOOPs.                *
* This illustrates the most straightforward way to perform a simulation.    However, the    *
* resulting dataset has I*N records (I=number of iterations, N=sample size per iteration).    *
* If I or N are large, the simulation might blow up.                             *
*********************************************************************          *;
DATA work.simulate;
   do iteration=1 to 30;
      do i=1 to 10;
         random=(rannor(1)*10) + 50;
         output;
      end;
   end;
run;
PROC PRINT data=work.simulate (firstobs=1 obs=31);
   format random 5.2;
   title 'Demonstration 3';
   title2 'First 31 records';
run;
PROC MEANS data=work.simulate max mean std maxdec=2 noprint;
   var random;
   by iteration;
   output out=work.max
            max=max;
   title 'Demonstration 3';
   title2 'Check the dataset';
run;
PROC PRINT data=work.max (firstobs=1 obs=30);
   format max 5.2;
   title 'Demonstration 3';
   title2 'Maxima by iteration';
run;
*********************************************************************          *
* This illustrates another way to perform the simulation -- which is to generate 1 record per    *
* iteration, with each simulated record generating a separate variable.    The number of    *
* records equals I, and the number of variables is N.    If N is large the number of variables    *
* might blow up the simulation.    Also, you would want to replace the 10s with N, in order    *
* make the code a bit more general.    The ARRAY statement creates 10 variables: X1          *
```

```
* through X10.  The DO loop works its way through those 10 variables.  X[i] is the syntax  *
* for referencing a variable within an array.  We will also encounter arrays when          *
* restructuring datasets.                                                                  *
*******************************************************************************           *;
DATA work.simulate;
   array x {*} x1-x10;
      do i=1 to 10;
         x[i]=(rannor(1)*10) + 50;
      end;
      max=max(of x1-x10);
run;

PROC PRINT data=work.simulate;
   format x1-x10 max 5.2;
   title 'Demonstration 4';
   title2 'Simulated dataset, 1 iteration';
run;
DATA work.simulate (keep=iterate max);
   array x {*} x1-x10;
      do iterate=1 to 30;
         do i=1 to 10;
            x[i]=(rannor(1)*10) + 50;
         end;
         max=max(of x1-x10);
         output;
      end;
   run;
PROC PRINT data=work.simulate;
   format max 5.2;
   title 'Demonstration 4';
   title2 'Maxima by iteration'
run;
*******************************************************************************           *;
* An approach which works for very large values of I and N is to make all the calculations on *
* the fly.  Here, we assume that the task is to place the maxima into bins (e.g., 40-49.99,  *
* 50-50.99, etc.).  So, at the end of each iteration, we add 1 to the appropriate bin.  The  *
* code could be written more efficiently -- this is just to illustrate the idea (using an ELSE-IF *
* syntax).  This executes 1 million iterations in a few seconds.                           *
*******************************************************************************           *;
```

```
************************************************************************      *
* Exercise: Diagram the code. The PUT statement illustrates one way to test the program      *
* logic.                                                                      *
************************************************************************      *;
DATA work.simulate;
      bin20=0;
      bin30=0;
      bin40=0;
      bin50=0;
      bin60=0;
      bin70=0;
      bin80=0;
      bin90=0;
      bin100=0;
      do iterate=1 to 30;
         do i=1 to 10;
            random=(rannor(1)*10) + 50;
            if i=1 then maximum=random;
            else maximum=max(random,maximum);
            file log;
            if (iterate=1 or iterate=2) then put _all_;
         end;
      if (maximum<30) then bin20=bin20+1;
      else if (30 <= maximum <40) then bin30=bin30+1;
      else if (40 <= maximum <50) then bin40=bin40+1;
      else if (50 <= maximum <60) then bin50=bin50+1;
      else if (60 <= maximum <70) then bin60=bin60+1;
      else if (70 <= maximum <80) then bin70=bin70+1;
      else if (80 <= maximum <90) then bin80=bin80+1;
      else if (90 <= maximum <100) then bin90=bin90+1;
      else if (100 <= maximum) then bin100=bin100+1;
      end;
      output;
   run;
PROC PRINT data=work.simulate;
   var bin20--bin100;
   title 'Demonstration 5';
   title2 'Summary results, 30 iterations';
run;
```

Appendix 2c: Simulation Module SAS Output

Demonstration1

Use of a DO loop and a random variable function to write 10 records

Obs	i	random
1	1	68.05
2	2	49.20
3	3	53.97
4	4	39.17
5	5	72.38
6	6	43.76
7	7	55.14
8	8	49.13
9	9	44.06
10	10	50.32

Demonstration 2

Calculate and output the maximum

The MEANS Procedure

Analysis Variable: random

Maximum	Mean	Std Dev
72.38	52.52	10.53

Demonstration 2

Verify that the maximum value has been written to a new dataset

Obs	_TYPE_	_FREQ_	max
1	0	10	72.38

Demonstration 2

Simulated data, forgetting the output statement

Obs	i	random
1	11	50.32

Demonstration 3

First 31 records

Obs	iteration	i	random
1	1	1	68.05
2	1	2	49.20
3	1	3	53.97
4	1	4	39.17
5	1	5	72.38
6	1	6	43.76
7	1	7	55.14
8	1	8	49.13
9	1	9	44.06
10	1	10	50.32
11	2	1	42.62
12	2	2	47.50
13	2	3	56.85
14	2	4	41.96
15	2	5	42.56
16	2	6	42.04
17	2	7	53.41
18	2	8	46.99
19	2	9	36.50
20	2	10	54.33
21	3	1	63.06
22	3	2	64.25
23	3	3	45.84
24	3	4	66.14
25	3	5	39.42
26	3	6	40.52
27	3	7	59.54
28	3	8	53.92
29	3	9	49.24
30	3	10	62.21
31	4	1	43.69

Demonstration 3

Maxima by iteration

Obs	iteration	_TYPE_	_FREQ_	max
1	1	0	10	72.38
2	2	0	10	56.85
3	3	0	10	66.14
4	4	0	10	61.84
5	5	0	10	62.77
6	6	0	10	69.28
7	7	0	10	66.06
8	8	0	10	63.80
9	9	0	10	63.79
10	10	0	10	65.00
11	11	0	10	56.99
12	12	0	10	62.38
13	13	0	10	57.71
14	14	0	10	63.75
15	15	0	10	67.40
16	16	0	10	66.19
17	17	0	10	55.91
18	18	0	10	64.20
19	19	0	10	82.44
20	20	0	10	71.02
21	21	0	10	70.45
22	22	0	10	69.89
23	23	0	10	65.63
24	24	0	10	67.11
25	25	0	10	60.95
26	26	0	10	56.84
27	27	0	10	57.94
28	28	0	10	64.16
29	29	0	10	60.87
30	30	0	10	73.10

Demonstration 4

Simulated dataset, 1 iteration

Obs	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	i	max
1	68.05	49.20	53.97	39.17	72.38	43.76	55.14	49.13	44.06	50.32	11	72.38

Demonstration 4

Maxima by iteration

Obs	iterate	max
1	1	72.38
2	2	56.85
3	3	66.14
4	4	61.84
5	5	62.77
6	6	69.28
7	7	66.06
8	8	63.80
9	9	63.79
10	10	65.00
11	11	56.99
12	12	62.38
13	13	57.71
14	14	63.75
15	15	67.40
16	16	66.19
17	17	55.91
18	18	64.20
19	19	82.44
20	20	71.02
21	21	70.45
22	22	69.89
23	23	65.63
24	24	67.11
25	25	60.95
26	26	56.84
27	27	57.94
28	28	64.16
29	29	60.87
30	30	73.10

Demonstration 5

Summary results, 30 iterations

Obs	bin20	bin30	bin40	bin50	bin60	bin70	bin80	bin90	bin100
1	0	0	0	6	19	4	1	0	0