A Curriculum Review of Programming Courses in a Master of Biostatistics Program

Jesse D. Troy[1,*], Megan L. Neely[1], Steven C. Grambow[1], Gina-Maria Pomann[1] & Gregory P. Samsa[1]

[1]Department of Biostatistics and Bioinformatics, School of Medicine, Duke University, Durham, NC, USA

*Correspondence: Department of Biostatistics and Bioinformatics, School of Medicine, Duke University, Durham, NC, USA. Tel: 1-919-668-2932. E-mail: jesse.troy@duke.edu

Received: September 25, 2023 Accepted: November 15, 2023 Online Published: February 15, 2024

doi:10.5430/jct.v13n1p405 URL: https://doi.org/10.5430/jct.v13n1p405

Abstract

Computing is a core competency for collaborative biostatisticians, who work on interdisciplinary scientific teams in medicine and publich health. However, computing is a broad field that encompases many underlying pedagogical constructs and subspeciality topics, not all of which are relevant for practicing biostatisticians. Furthermore, the ubiquitous nature of computing in biostatistics and the variation in computing education required to support students' interests in biostatistics subspecialities presents a challenge for curriculum design. Specifically: where and how to integrate training in computing into biostatistics educational programs? We discuss here our answer to these questions as it relates to a 2-year, full-time, intensive master's degree program in biostatistics. Specifically, we define and rationalize the core pedagogical contstruct that guided our curriculum deisgn—computational thinking—and then discuss our two-fold approach to training biostatistics master's students in computing: creation of a 2-course series of targeted training in computing, and embedding training in computing throughout other courses in the curriculum. A detailed description of the course design is included along with a description of our instructional methods, including how we evaluate student performance.

Keywords: computational thinking, curriculum design, statistics, programming

1. Introduction

Collaborative biostatistics is an interdisciplinary specialty within the broader field of applied statistical practice (Begg & Vaughan, 2011; Pomann et al., 2020; Samsa, 2018). Collaborative biostatisticians work on a broad range of problems in medicine and public health, including drug development (e.g., laboratory research and human clinical trials), studies of disease etiology, health disparities, and more. The work of a collaborative biostatistician involves research study design and implementation, data analysis, and interpretation of results (Pomann et al., 2020, 2022; Samsa et al., 2018; Troy et al., 2021). Entry into the field typically requires a master's degree (O*Net OnLine, n.d.). Master's level biostatisticians usually work under the direction of a PhD biostatistician but are often given autonomy to make decisions around routine tasks, and even manage other more junior staff. Collaborative biostatisticians are found in industry settings such as pharmaceutical companies and contract research organizations (CROs), or at academic medical centers and schools of public health, in government and in the non-profit sector.

Successful collaborative biostatisticians are trained in several core competencies, some of which fall under the umbrella of computing (Pomann et al., 2020). Computing is broadly defined as "the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation and application" (P.J. Denning et al., 1989). Practical training for collaborative biostatisticians emphasizes computational thinking, which is a general problem-solving approach that includes, but is not limited to, writing computer programs (Peter J. Denning & Tedre, 2022). Computational thinking can be conceptualized as a set of interrelated skills and competencies: for example, abstraction, algorithms, and decomposition of large problems into smaller steps. In the context of data analysis computational thinking involves abstracting a problem, representing a solution using data elements, and then organizing and processing those data elements (Peter & Tedre, 2022).

The importance of rigorous training in computing, including computational thinking, as part of general statistics education curricula has been recognized for some time (Nolan & Temple Lang, 2010). In fact, modern applied

statistical practice is essentially impossible without computing. In recognition of this, some authors have emphasized the need to develop targeted training programs in 'statistical computing' that would effectively elevate this competency to its own sub-specialty of statistical practice (Hardin et al., 2021). This emphasis is driven largely by the increasingly more common integration of statisticians into teams that develop software which is used repeatedly across several projects by business or scientific teams, thus necessitating training in principles of software engineering, data structures, and algorithms not to mention modern work practices around literate and reproducible programming, source code revision control, and more (Reinhart & Genovese, 2021).

Our focus on computing in statistics is narrower. Specifically, we have created a 2-year, intensive master's degree program in biostatistics that prepares students to enter an interdisciplinary scientific workforce where the majority of their application of computing is for single-use purposes, e.g., specific to a pre-planned statistical analysis for a single research study rather than targeted at development of a product intended for repeated use to solve a broader class of problems (Neely et al., 2022; Troy, Granek, et al., 2022; Troy, McCormack, et al., 2022). In fact, the majority of our graduates enter the workforce in traditional clinical research roles in industry, while a smaller proportion join academic medical centers yet still focus on clinical research. Although this is the majority of our graduates, a small but non-trivial proportion of our students seek work with "big data" in the health-related context, e.g., processing and analyzing large electronic medical records datasets. Although these students are the smallest proportion of our trainees, they are the most likely to benefit from more advanced training in computing.

From a curriculum design perspective our challenge has been to target our training in computing appropriately to student interests and needs, and to successfully integrate this training within our broader curriculum, e.g., by not creating a silo of computational training that is entirely separate from training in research study design, statistical methods, and the like. While all of our students require a strong foundation in computing, some require a stronger emphasis on specific things than others. For example, while we agree with the current opinion on education of biostatisticians in basic principles of software engineering, it does little service to our students who want a career in clinical trials to ask that they learn how to program data structures like heaps. While everyone requires training in algorithms and data structures at some level, a heap is likely only to be encountered by our students interested in a class of problems related to big data in healthcare. Work on clinical trials typically doesn't require this depth of knowledge in computing. Thus, our emphasis was not on creating a single course to teach computing in biostatistics but rather to integrate education in computing throughout the curriculum in such a way that it targets the career development goals of our students. Our solution was to create a 2-semester course sequence in computing that has been designed to integrate with other courses in the curriculum, providing flexibility by allowing students a choice of classes based on their interests.

We describe here the structure of our master's degree program as well as the underlying pedagogical constructs we considered when designing the statistical programming course sequence. This is followed by a statement of our pedagogical goals for the programming courses and an overview of the course design. We then briefly discuss how instruction in programming is embedded throughout the other courses in our curriculum. Finally, we close with a discussion of our instructional methods and how students are evaluated.

2. Coursework in the Master of Biostatistics Program

Students in our master of biostatistics program complete their coursework over a two-year period. Courses are organized into sequences during the first year, in which students receive instruction in the core curriculum areas of the practice of biostatistics, applied data analysis, statistical theory, career preparation, and statistical programming with some allowance for electives within these sequences based on undergraduate preparation and student's future career or academic interests (Troy, McCormack, et al., 2022). During their second year, students pursue elective courses in their chosen specialty: biomedical data science, clinical and translational research, or mathematical statistics. Graduates who select any of these specialty areas might either pursue employment or further graduate education, usually in PhD programs in biostatistics or related fields. In the following sections we describe the design of our first-year programming course sequence and its emphasis on computational thinking. We also describe a general framework for how we have integrated continued instruction in computational thinking throughout the program courses.

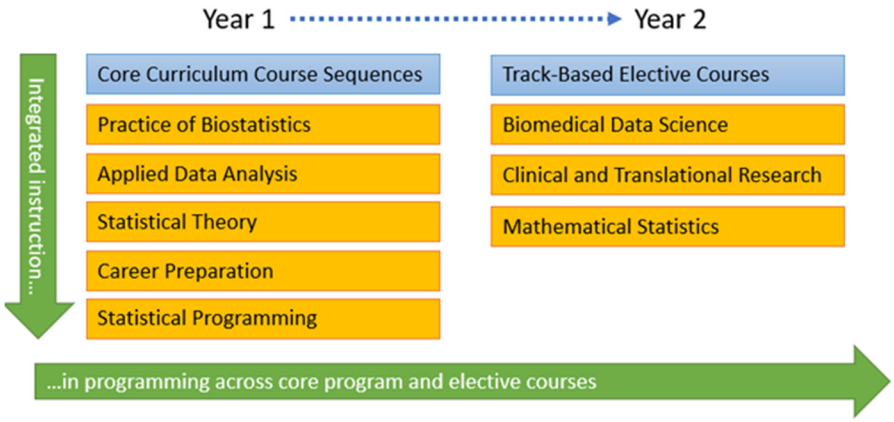

Figure 1. Integration of Instruction in Programming Throughout the Master of Biostatistics Courses

3. Pedagogical Constructs Related to Curriculum Design for Programming Courses

To encourage precision in language, it is helpful to distinguish computing, computational thinking and programming. Computing is an academic discipline that overlaps with computational thinking (P.J. Denning et al., 1989; Peter J. Denning & Tedre, 2022). In contrast to much of the public discourse around computational thinking in education, which focuses on the basics of computational thinking for children and other novice learners,(Buitrago Flórez et al., 2017; Cortina, 2007; Wing, 2006) computing includes both basic and advanced computational thinking, is informed by mathematics and engineering, among others, and is concerned generally with the question of whether and how tasks can be efficiently automated (P.J. Denning et al., 1989).

Computational thinking is a general problem-solving approach that includes, but is not limited to, writing computer programs. Indeed, Denning proposes that "Computational thinking is the mental skills and practices for designing computations that get computers to do jobs for us, and for explaining and interpreting the world in terms of information processes" (Peter J. Denning & Tedre, 2022). Computational thinking can be conceptualized as a set of interrelated skills and competencies: for example, abstraction, algorithms, and decomposition of large problems into smaller steps. In the context of data analysis computational thinking involves abstracting a problem, representing a solution using data elements, and then organizing and processing those data elements (Peter J. Denning & Tedre, 2022).

Programming reifies and instantiates computational thinking, and, indeed, most discourse about computational thinking pertains not to algorithms in the abstract but, instead, to how to implement them *via* computer code. As applied to programming, five key aspects of computational thinking are modularity, data structures, encapsulation, control structures, and recursion, which we refer to as "algorithms and data structures" (Peter J. Denning & Tedre, 2022). In contrast to a narrow definition of programming that focuses solely on language syntax, our usage of the term ‘programming’ includes broader concepts derived from computational thinking, encompassing algorithms, data structures, program design, reproducibility and other related concepts.

3.1 Illustration of the Interface Between Computational Thinking and Programming in Biostatistics

To illustrate some relationships between computational thinking and programming, (Buitrago Flórez et al., 2017) consider the simple task of calculating the sum of the squares of each of the integers from 1 to 5, and doing so using the programming language of SAS. One possible solution is the code in Appendix A (which is written to be human-friendly rather than machine-efficient). The output dataset produced by this program is shown in Table 1.

Description: This table shows the output of the program shown in Appendix A that computes the sum of the squares of the first five integers. The Observation column simply numbers the rows of the table; the N column shows the highest integer for which the program is computing the sum of squares; column I shows the integer being squared; the I_SQ column shows the square of the integer; and the COUNTER sums the values of I_SQ across the rows (where the value of COUNTER is set equal to I_SQ on the first row).

Table 1. Output from a Program That Computes the Sum of the Squares of the First Five Integers

Observation	N	I	I_SQ	COUNTER
1	5	1	1	1
2	5	2	4	5
3	5	3	9	14
4	5	4	16	30
5	5	5	25	55

The most concrete part of programming is the syntax of the SAS code. For example, all SAS statements end with a semicolon. One element of programming in a specific language is learning its syntax. Another element of programming is the use of sound coding practices. In this example, the following sound practices can be observed:

- Extensive comments are included.
- The comments explicitly describe the algorithm used to accomplish the programming task.
- Indentation and white spaces are used to enhance readability.
- Output is titled.
- The program is written in some level of generality -- for example, instead of solving the problem of writing the sum of squares of 5 integers, it solves the problem of writing the sum of n integers, with n=5 as a use case.

Elements of computational thinking used in designing the above program include the related constructs of algorithms and data structures. As the program is being designed, the programmer may visualize the above output dataset and then the algorithm which generates it, or visualizes the algorithm and recognizes that it generates the output dataset, or visualize both simultaneously. For our purposes: *Computational thinking tells the student what to do, and programming skills tell the student how to do it well*. Moreover, those elements of programming skills specific to SAS tell the student how to do it well in SAS.

4. Pedagogic Goals for the Programming Courses

Our statement of overall pedagogic goals was based on four premises. First, elements of computational thinking form much of the basis for developing general programming skills -- which we operationally define as a combination of sound programming practices and facility with algorithms and data structures. Second, since our students are training to use programming as a tool, (i.e., in contrast to a computer scientist who might study programming languages from a theoretical perspective (Chomsky, 1956; Jäger & Rogers, 2012), instruction should focus on solving specific tasks using specific languages. For example, complex statistical simulations could be programmed in SAS using loops, or in R using vectorized “apply” functions. Third, all else being equal, the programming languages we teach should correspond to those desired by doctoral programs and employers. Finally, instruction about programming should be formalized and embedded throughout the curriculum rather than limited to programming-related courses.

Thus, our overall pedagogic goals can be summarized as: *To* provide *explicit instruction on general programming principles, algorithms, and data structures within the framework of specific tasks, utilizing contemporary programming languages and seamlessly integrating this instruction throughout the curriculum.* Implicit in this statement of pedagogic goals is that we are teaching programming as programming, which by nature includes principles from computational thinking, but also that we are deemphasizing other aspects of computing that receive considerably more attention within other (not biostatistics) areas of the academy, like computer science (P.J. Denning et al., 1989; Peter J. Denning & Tedre, 2022; Reinhart & Genovese, 2021).

5. Design of the Computing Course Sequence

With the above goal in mind, our formal training in programming begins by illustrating basic programming techniques (e.g., creation of datasets, filtering, conditional flow of control, using functions to break large tasks into smaller ones) using the language of R as the use case. The initial exposure to R occurs during a pre-orientation curriculum taken in the summer before year 1, which ensures that all entering students can successfully interact with its interface (Neely et al., 2022).

In their second semester, students select one of two courses that teach algorithms and data structures. The first option

is a course in Python that students are encouraged to take if they are interested in research related to big data applications in medicine and public health. Students interested in traditional career pathways in clinical research typically select the SAS course since SAS remains the *de facto* standard at most contract research organizations and pharmaceutical companies, which are the primary industry employers of our graduates.

The Python course also includes elements which are partially independent of language, e.g., covering data structures like stacks, queues, linked lists, and trees. In contrast, the SAS course uses task-based instruction, with students practicing common data management and analysis tasks. General programming principles introduced in the pre-requisite R course are also illustrated. Because SAS and R use different data structures, algorithms and data structures are introduced *via* comparison (e.g., similarities and differences between accomplishing a task in SAS versus R).

Some might consider this curriculum "programming-heavy", as it devotes two semesters to programming rather than statistical methodology. We made this decision, despite knowing students can easily learn the language syntax independently. However, the courses cover more than simply language syntax and emphasize general programming principles. The ultimate goal of this “programming heavy” curriculum is: *to develop excellent programmers who write literate, reproducible, efficient, and human-friendly programs and can continue to improve their programming skills throughout their careers.*

6. Embedding Progamming Content in Other Courses

Three types of programming content are integrated within our non-programming courses: (1) data management; (2) standard statistical analysis; and (3) computationally-oriented statistical analysis. Regarding the first item, data management is practiced within internships, the masters' project, and similar applications. Formal instruction in language syntax pertaining to data management is provided within the computing course sequence (e.g., how to merge/join datasets in R and SAS). Less language-specific instruction about building reproducible data management systems is contained within a course sequence in statistical practice.

The second type of programming content, standard statistical analysis, essentially involves manipulating raw data into an analysis dataset, applying an existing analysis function to the analysis dataset, passing information to that function (e.g., specifying the predictor and outcome variables in a regression model), capturing elements of that function's output, and then either printing or performing additional manipulations on the output. In the computing courses we provide illustrations of this process. For example, we describe the basic syntax by which it can be accomplished in R and SAS, including the name and structure of the output objects, etc. Our non-programming courses assume that students are able to follow this process, and focus on the details of applying the analysis functions from R and SAS that are relevant to the specific methods of statistical analysis taught in each course.

Appendix B illustrates how the third type of programming content, computationally-oriented statistical programming, is integrated into our first statistical inference course. As a summary of the example, students are presented with results of a hypothetical 2-arm clinical trial with 10 patients per arm, in which the outcome measure is a 5-level ordinal variable. The students are then asked to generate a sampling distribution under the null hypothesis. Students should recognize this as an urn problem in which a key step is to determine the number of ways to draw 10 balls out of an urn containing 20 balls with 5 different colors. The statistical specification of the problem can be translated into a data structure with one row for each possible way to draw the 10 balls. For each row, a calculation is made, and a new variable (i.e., the "distance metric" which quantifies how different the two groups are) is generated. Finally, a frequency distribution of the distance metric (i.e., summarizing the results over the rows) becomes the basis for the statistical test.

Within this overall algorithm, students must solve the technical problem of enumerating all the possible ways to draw the 10 balls. A direct approach is to create a list of all the binary numbers from '00000000000000000000' through '00000000000000000001' to '11111111111111111111', filter in those with exactly ten 1's, and then map these binary numbers to the balls in the urn. For example, '11111111110000000000' would map to selecting the balls labeled 1 through 10. A clever approach is finding an R function that accomplishes this task or something sufficiently similar, which can then be repurposed.

To comment: (1) to teach statistical inference, a key step involves translating a statistical procedure into an algorithm and data structure; (2) the above algorithm is independent of language, although we expect that students will implement it in R, since they have already been exposed to that language; and (3) the programming task is sufficiently substantial to also provide a use case for additional instruction about how to develop and test a program,

concepts that are also introduced in the formal programming courses.

Appendix C illustrates how data management and statistical programming is integrated into a first-year course in regression. The task is to perform a model validation by breaking a dataset into training and test datasets, fitting a linear regression on the training dataset, and then fitting the same model (i.e., with the parameters from the training dataset) to the test dataset. The resulting file contains the observed and fitted values of the outcome variable within the test dataset, which then become the inputs to a validation analysis. The task illustrates the sequence of programming steps by printing the datasets at each point in the process. In essence, this is a case where the programmer visualizes the sequence of output datasets, which then implies an algorithm. The final step in the process is discovering the SAS code needed to implement the algorithm.

7. Instructional Methods and Evaluation

In our curriculum, programming is primarily taught within the context of accomplishing specific tasks using specific languages. The general approach, consistent with principles of active learning, is "see a simple example, do a simple example, reflect on the experience, do increasingly more challenging examples, reflect, etc.". Much of the work is performed in groups, as this facilitates learning and is consistent with how most of our graduates will practice biostatistics.

Evaluation includes both process and outcome. The outcome (i.e., work product) is the program the student has written. The evaluation of outcome includes the degree to which the program accomplished its objectives, for example, whether elements of the program pass unit testing and whether the program as a whole produces the correct outputs for a sufficiently extensive set of inputs (Russell et al., 2019). The outcome evaluation also includes the degree to which the program follows general principles of literate, reproducible and human-friendly programming (*(218) Knuth on Literate Programming - YouTube*, n.d.; Baker, 2016; Knuth, 1984).

Another element of instruction and evaluation is the process by which students design and test their code. Our experience is that many students address a programming task by immediately diving in and writing code, a suboptimal approach. Students should instead follow a standard process when developing code that includes robust and reproducible testing and debugging. This process should include an explicit problem specification, a description of the algorithm, and a procedure for testing. In some cases, the problem specification could be a single sentence recapitulating their understanding of the task (e.g., divide the data set into training and test samples, fit a logistic regression on the training sample, and validate the results on the test sample). In other cases, the specification will be more detailed, as illustrated in the example pertaining to statistical inference. The algorithm could be described in words, a flowchart, pseudocode, a sequence of illustrative data files, etc. An element of the testing procedure could be unit testing, with specific criteria for success. Another element could be tracing the code (e.g., printing intermediate results as the program is being developed, printing sample records for each step in the data management sequence, etc.).

A form of evaluation used in the SAS course is a coding interview, whereby students describe how they would solve various programming problems using either SAS code or pseudocode. As would be the case for a coding interview conducted by a potential employer, the primary assessment criteria pertain to work processes, for example, clearly describing an algorithm, specifying a testing procedure, etc. The more correctly the problem is solved and the more sophisticated the approach to doing so, the better. This evaluation method produces informative results and is typically well-liked by students (Samsa, 2020).

The evaluation of the programming sequence is based on two criteria, derived from our two main educational goals. First, based on a review of course content, our approach focuses on general programming principles, algorithms and data structures within the context of accomplishing specific tasks using the languages of R, SAS, and Python. Thus, the first goal is substantially met. Second, our assessment of our students' programming skills is based on examinations in the programming sequence classes and the degree to which students can successfully apply their skills in their non-programming (e.g., statistics) classes. As an example of the latter, a second-year instructor chose to use a programming language, Stata, which is outside the scope of our core programming instruction. The instructor could reasonably assume that, as excellent programmers familiar with general programming principles, our students could develop tourist-level functionality in this new language without undue difficulty.

8. Summary

We reported here on our efforts to integrate targeted instruction in computing, especially computer programming

with an emphasis on computational thinking, into a short-term, 2-year intensive master's degree program in biostatistics that services primarily job-bound students seeking work in medicine and public health research. Clarifying terminology and educational objectives proved to be a critical first step. Our faculty are diverse and "computation" doesn't necessarily mean the same thing to everyone. Ultimately, we realized that the core construct which we were discussing was programming, something which everyone understood similarly. This recognition led to the development of two related educational goals: (1) *To* provide *explicit instruction on general programming principles, algorithms, and data structures within the framework of specific tasks, utilizing contemporary programming languages and seamlessly integrating this instruction throughout the curriculum*; in order to (2) *develop excellent programmers, who write literate, reproducible and human-friendly programs, and can continue to improve their programming skills throughout their careers.* Here, programming is defined broadly rather than narrowly, and incorporates principles of computational thinking. In essence, we want students to consider programming to be a tool which they can confidently apply in multiple contexts.

9. Conclusion

Previous literature about teaching computational thinking (Buitrago Flórez et al., 2017; Cortina, 2007; Wing, 2006) applies to students in non-science-related fields, children and other novice learners. Although this literature offers some relevant insights, such as the benefit of active engagement, we consider our target audience to be fundamentally different. Their mathematical training implies familiarity with abstraction, decomposition of large problems into smaller constituent parts, algorithms, etc. (Peter J. Denning & Tedre, 2022). In essence, we can rely on our students to already be comfortable with the basics of computational thinking, and thus our task is not to teach computational thinking *de novo* but to help students translate their existing computational thinking skills into the context of programming.

Another critical step in our curriculum review was the decision to focus on the programming skills required of all students, regardless of career focus. Thus, the non-trivial question of what additional programming training requirements might be needed for students interested in big data applications in medicine and public health was deferred (Nolan & Temple Lang, 2010; Schwab-McCoy et al., 2021). We would also like to highlight the importance of domain knowledge in algorithm development and, thus, in programming (Peter J. Denning & Tedre, 2022). Our practice of biostatistics sequence (Troy, Granek, et al., 2022) provides students with practice in learning scientific content, following the premise that understanding this content is necessary for correctly embedding it within a statistical framework and model. Task-based instruction in programming follows a similar belief: students need not master extensive details of a programming language (as would be the case in training for formal certification) but should instead focus on tasks they would typically accomplish. Ideally, instruction should combine what to do (e.g., how to design a statistically sound method of validating a regression model) with how to do it (e.g., how to program the implementation, as per Appendix 3).

The most actionable result of our curriculum review was the decision to integrate programming more intentionally within our biostatistical course offerings in order to incentivize our students to treat programming as a tool that they can confidently use to (among others) visualize data, manage data, analyze data, perform simulations, and develop statistical intuition.

In closing, we found the process of formalizing and reconsidering the rationale for our instruction in programming to be a helpful exercise, which we recommend to others.

References

(218) Knuth on Literate Programming - YouTube. (n.d.). Retrieved March 31, 2023, from https://www.youtube.com/watch?v=Mr3WTR0a5SM

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*. https://doi.org/10.1038/533452a

Begg, M. D., & Vaughan, R. D. (2011). Are biostatistics students prepared to succeed in the era of interdisciplinary science? (and how will we know?). *American Statistician*, *65*(2), 71-79. https://doi.org/10.1198/tast.2011.10222

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research* , *87*(4), 834-860. https://doi.org/10.3102/0034654317710096

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, *2*(3), 113-124. https://doi.org/10.1109/TIT.1956.1056813

Cortina, T. J. (2007). An introduction to computer science for non-majors using principles of computation. *SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education*, 218-222. https://doi.org/10.1145/1227310.1227387

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

Denning, Peter J., & Tedre, M. (2022). Computational Thinking: A Disciplinary Perspective. *Informatics in Education*, *20*(3), 361-390. https://doi.org/10.15388/INFEDU.2021.21

Hardin, J., Horton, N. J., Nolan, D., & Lang, D. T. (2021). Computing in the Statistics Curricula: A 10-Year Retrospective. *Journal of Statistics and Data Science Education*, *29*(S1), S4-S6. https://doi.org/10.1080/10691898.2020.1862609

Jäger, G., & Rogers, J. (2012). Formal language theory: refining the Chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *367*(1598), 1956. https://doi.org/10.1098/RSTB.2012.0077

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), 77-85. https://doi.org/10.5430/jct.v11n4p77

Nolan, D., & Temple Lang, D. (2010). Computing in the Statistics Curricula. *The American Statistician*, *64*(2), 97-107. https://doi.org/10.1198/tast.2010.09132

O*Net OnLine. (n.d.). *Biostatisticians*. Retrieved March 30, 2023, from https://www.onetonline.org/link/summary/15-2041.01#Education

Pomann, G.-M., Boulware, L. E., Cayetano, S. M., Desai, M., Enders, F. T., Gallis, J. A., Gelfond, J., Grambow, S. C., Hanlon, A. L., Hendrix, A., Kulkarni, P., Lapidus, J., Lee, H.-J., Mahnken, J. D., McKeel, J. P., Moen, R., Oster, R. A., Peskoe, S., Samsa, G., … Thomas, S. M. (2020). Methods for training collaborative biostatisticians. *Journal of Clinical and Translational Science*. https://doi.org/10.1017/cts.2020.518

Pomann, G.-M., Boulware, L. E., Chan, C., Grambow, S. C., Hanlon, A. L., Neely, M. L., Peskoe, S. B., Samsa, G., Troy, J. D., Yang, L. Z., & Thomas, S. M. (2022). Experiential Learning Methods for Biostatistics Students: A Model for Embedding Student Interns in Academic Health Centers. *Stat*, *11*(1), e506. https://doi.org/10.1002/sta4.506

Reinhart, A., & Genovese, C. R. (2021). Expanding the Scope of Statistical Computing: Training Statisticians to Be Software Engineers. *Journal of Statistics and Data Science Education*, *29*(S1), S7-S15. https://doi.org/10.1080/10691898.2020.1845109

Russell, S., Bennett, T. D., & Ghosh, D. (2019). Software engineering principles to improve quality and performance of R software. *PeerJ Computer Science*, *2019*(2). https://doi.org/10.7717/PEERJ-CS.175/SUPP-4

Samsa, G. P. (2018). 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. (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

Schwab-McCoy, A., Baker, C. M., & Gasper, R. E. (2021). Data Science in 2020: Computing, Curricula, and Challenges for the Next 10 Years. *Journal of Statistics and Data Science Education*, *29*(S1), S40-S50. https://doi.org/10.1080/10691898.2020.1851159

Troy, J. D., Granek, J., Samsa, G. P., Pomann, G.-M., Updike, S., Grambow, S. C., & Neely, M. L. (2022). 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., McCormack, K., Grambow, S. C., Pomann, G.-M., & Samsa, G. P. (2022). 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

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215

Appendix A

SAS Code to Calculate the Sums of Squares of Integers from 1-5

The following program code was written and executed on SAS 9.4 (SAS Institute, Cary, NC) on a computer running Microsoft Windows 10 64-bit Enterprise Edition.

```
************************************************************
* Calculate the sum of the squares of n integers.
* Initialize a counter variable containing the result to 0.
* Then, loop over the integers from 1 to n, indexed by i.
* At each point in the loop, create a new variable containing
* the square of i.  Add this variable to the counter.
* Write the results.
************************************************************;

* Create a SAS dataset containing the results;
data temp;

* Initialize the counter;
  retain counter 0;

* Specify the largest integer in the sum (5 in this example);
  n=5;

* Loop over the integers from 1 to n;
    do i=1 to n;

* Square i;
      i_sq=i*i;

* Add the square of i to the counter;
      counter=counter+i_sq;

* Write the result;
      output temp;

* Close the do loop;
  end;
```

```
* End the data step;
  run;

* Print the results;
proc print data=temp;
  var n i i_sq counter;
  title 'sum of squares of n integers';
  title2 'n=5';
run;
```

Appendix B

Illustration of How Computationally-Oriented Statistical Programming is Integrated into a First Course on Statistical Inference

Table 1: Pilot study results

Group	Terrible (0)	Poor (40)	Fair (50)	Good (80)	Excellent (100)
A	2	2	2	2	2
B	1	1	1	3	4

Table 1 presents the results of a pilot randomized clinical trial comparing two interventions: A and B. The outcome variable, health status, is ordinal -- for example, terrible is the worst, and excellent is the best. The categories don't necessarily represent equal intervals -- for example, there's a wide gap between terrible and poor and relatively little difference between poor and fair. The numbers associated with the categories are based on the investigator's clinical judgment about the meaning of the categories.

The dataset is small enough to be able to perform exact inference. Apply the logic behind a Fisher's exact test to generate an exact test to test the null hypothesis that the 2 groups have identical outcomes. In other words, you'll need to condition on the margins of the table: 10 patients in group A, 10 patients in group B, 3 patients in terrible health, 3 patients in poor health, 3 patients in fair health, 5 patients in good health, and 6 patients in excellent health. An exact test is based on a sample space that includes all the possible arrangements of the data.

As you extend the Fisher's Exact Test, you'll discover that you need a distance metric to quantify how different the results are between the 2 groups. An ideal metric would be 0 if the results are identical and large when the results differ greatly. Use the 2 cumulative distribution functions as inputs to this metric -- the metric is the maximum distance between the two cumulative distribution functions, regardless of direction.

Table 2 illustrates calculating the distance metric for the pilot study -- its value is 0.30.

To perform the desired exact test, verify that the sample space consists of exactly 184,756 possible tables. Attach your R code. Under the null hypothesis, these 184,756 tables are all equally likely. Calculate an exact distribution of the test statistic under the null hypothesis. Use this distribution to compute an exact p-value.

Table 2: Calculating the distance metric

Group	Terrible (0)	Poor (40)	Fair (50)	Good (80)	Excellent (100)
A	2	2	2	2	2
B	1	1	1	3	4
Distance	2/10-1/10 = 0.10	4/10-2/10 = 0.20	6/10-3/10 = 0.30	8/10-6/10 = .20	10/10-10/10 = 0

Answer:

This can be treated as an urn problem, where the urn has 3 balls of color A, 3 balls of color B, 3 balls of color C, 5

balls of color D, and 6 balls of color E and, under the null hypothesis, the selection of balls is independent. Label the balls as X1-X20.

We need a way to enumerate all the possible ways to select 10 balls from the urn. The desired data structure will have one row for each possible way to make this selection.

One of these many rows will contain the raw data. Labelling the 5 categories in group A as A1-A5 and the 5 categories in group B by B1-B5, this row will contain:

Row number	*A1*	*A2*	*A3*	*A4*	*A5*	*B1*	*B2*	*B3*	*B4*	*B5*	*Distance metric*
#	*2*	*2*	*2*	*2*	*2*	*1*	*1*	*1*	*3*	*4*	*0.30*

Once we have A1-A5 and B1-B5, the calculation of the distance metric is straightforward and, indeed, is illustrated in table 2.

To get to the desired data structure, we need an intermediate data structure to (1) enumerate all the possible ways to draw the balls from the urn; and (2) translate the labels for those balls into A1 through B5. This can be accomplished in multiple ways. A direct, albeit not computationally efficient, way is to (1) create a list of all the binary numbers from 00000000000000000000 through 11111111111111111111; (2) filter in those numbers containing exactly 10 1's, such as 11111111100000000000; (3) map the pattern of 0s and 1s to the identity of the chosen balls; and (4) count the number of balls of each type.

To illustrate the last 2 steps, label the 20 balls as X1-X20, and assume that X1-X3 correspond to "terrible", X4-X6 correspond to "poor", X7-X9 correspond to "fair", X10-X14 correspond to "good", and X15-X20 correspond to "excellent". For example, the binary number 11111111110000000000 implies that X1-X10 were selected into group A, and so A1=3, A2=3, A3=3, A4=1, A5=0, B1=0, B2=0, B3=0, B4=4, B5=6.

The final step is to create a frequency table for the distance metric, where each row contributes a single observation. The result is presented below. The exact p-value is 1-.469 = .531.

Distance metric	*Frequency*	*Cumulative probability*
0.1	*21,600*	*11.7%*
0.2	*65,100*	*46.9%*
0.3	*49,750*	*73.8%*
0.4	*33,672*	*92.1%*
0.5	*10,230*	*97.6%*
0.6	*3,392*	*99.4%*
0.7	*990*	*99.9%*
0.9	*22*	*100%*

Appendix C

An Example of Model Validation

The following program code was written and executed on SAS 9.4 (SAS Institute, Cary, NC) on a computer running Microsoft Windows 10 64-bit Enterprise Edition.

```
************************************************************************
* Randomly divide a dataset into training and test sets.  Run a linear
* regression on the training set, apply the regression parameters from
* the training set to the test set, and save the observed and predicted
* values from the test set as inputs into a model validation analysis.
```

```
*************************************************************************;

* Create the input file;
* X is the predictor, Y is the outcome;
data set1;
  input x y;
datalines;
1  2.4
2  4.2
3  6.9
4  9.3
5  9.8
6  10.6
7  12.8
8  16.0
9  16.1
10 18.7
;
run;

* Print the input dataset;
proc print dat6a=set1;
  title 'original input dataset: set1';
run;

* Add a new column containing a uniform pseudorandom variable*;
* The streaminit function explicitly sets a seed, allowing the results to be
reproduced;
* The rand() function performs the randomization;

data set2;
  set set1;
  call streaminit(1);
  rand_unif=rand("uniform",0,1);
run;

* Print the next dataset;
proc print data=set2;
  title 'next dataset: set2';
run;

* Create a new variable, rand_group, which assigns the smallest variables to one
```

```
group and the largest random variables to the other;

proc rank data=set2 out=set3 groups=2;
  var rand_unif;
  ranks new_group;
run;

* Print the next dataset;
proc print data=set3;
  title 'next dataset: set3';
run;

* Create separate versions of Y by group;
* Drop the pseudorandom number and y;
data set4;
  set set3;
  if new_group=0 then y0=y; else y0=.;
  if new_group=1 then y1=y; else y1=.;
  drop rand_unif y;
run;


* Print the next dataset;
proc print data=set4;
  title 'next dataset: set4';
run;

* Run the regression on group 0 but save predicted values for both groups;
proc reg noprint data=set4;
  model y0=x;
  output out=set5
        predicted=y_hat;
run;


* Print the next dataset;
proc print data=set5;
  title 'next dataset: set5';
run;

* Retain values of Y1 and Y_HAT for group 1;
* Also retain the value of X, although it isn't needed for the validation analysis;
```

```
data set6;
  set set5;
  if new_group=1;
  keep x y1 y_hat;
run;


* Print the next dataset;
proc print data=set6;
  title 'final dataset: set6, input to the validation analysis';
run;
```

The following is the output generated by the program.

SET1: original dataset

X	Y
1	2.4
2	4.2
3	6.9
4	9.3
5	9.8
6	10.6
7	12.8
8	16.0
9	16.1
10	18.8

SET2: add a uniform random variable

X	Y	RAND_UNIF
1	2.4	0.88387
2	4.2	0.97382
3	6.9	0.50758
4	9.3	0.88694
5	9.8	0.68936
6	10.6	0.94325
7	12.8	0.92879
8	16.0	0.48766
9	16.1	0.77989
10	18.8	0.87714

SET3: use the uniform random variable to define the training and test sets

X	Y	RAND_UNIF	NEW_GROUP
1	2.4	0.88387	1
2	4.2	0.97382	1
3	6.9	0.50758	0
4	9.3	0.88694	1
5	9.8	0.68936	0
6	10.6	0.94325	1
7	12.8	0.92879	1
8	16.0	0.48766	0
9	16.1	0.77989	0
10	18.8	0.87714	0

SET4: create different versions of the outcome Y by group

X	Y	RAND_UNIF	NEW_GROUP	Y0	Y1
1	2.4	0.88387	1	.	2.4
2	4.2	0.97382	1	.	4.2
3	6.9	0.50758	0	6.9	.
4	9.3	0.88694	1	.	9.3
5	9.8	0.68936	0	9.8	.
6	10.6	0.94325	1	.	10.6
7	12.8	0.92879	1	.	12.8
8	16.0	0.48766	0	16.0	.
9	16.1	0.77989	0	16.1	.
10	18.8	0.87714	0	18.8	.

SET5: run the regression using Y0 as the outcome and generate predicted values -- this works because group 1 is ignored when estimating the regression parameters but predicted values are generated for all observations, including group 1

X	Y	RAND_UNIF	NEW_GROUP	Y0	Y1	Y_HAT
1	2.4	0.88387	1	.	2.4	3.4235
2	4.2	0.97382	1	.	4.2	5.1029
3	6.9	0.50758	0	6.9	.	6.7824
4	9.3	0.88694	1	.	9.3	8.4618
5	9.8	0.68936	0	9.8	.	10.1412
6	10.6	0.94325	1	.	10.6	11.8206
7	12.8	0.92879	1	.	12.8	13.5000
8	16.0	0.48766	0	16.0	.	15.1794
6	16.1	0.77989	0	16.1	.	16.8588
10	18.8	0.87714	0	18.8	.	18.5382

SET6: filter in the observations from group 1, retain the predictors, the outcome, and the predicted outcome

X	Y1	Y_HAT
1	2.4	3.4235
2	4.2	5.1029
4	9.3	8.4618
6	10.6	10.1412
7	12.8	13.5000

Acknowledgments

Not applicable.

Authors contributions

Not applicable.

Funding

Not applicable.

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.