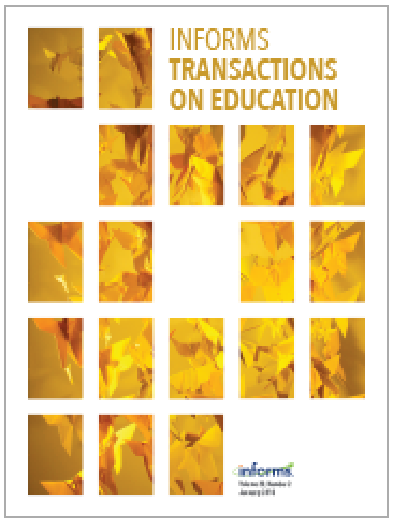


INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Interactive Computing for Accelerated Learning in Computation and Data Science

David L. Alderson

http://pubsonline.informs.org/journal/ited

INFORMS TRANSACTIONS ON EDUCATION
Articles in Advance, pp. 1–16
ISSN 1532-0545 (online)

Interactive Computing for Accelerated Learning in Computation and Data Science

David L. Alderson[a]

[a] Operations Research Department, Naval Postgraduate School, Monterey, California 93943
Contact: dlalders@nps.edu, https://orcid.org/0000-0002-1641-5302 (DLA)

Received: February 28, 2020
Revised: April 10, 2021; July 14, 2021
Accepted: July 26, 2021
Published Online in Articles in Advance: October 5, 2021

https://doi.org/10.1287/ited.2021.0261

Abstract. This article describes the motivation and design for introductory coursework in computation aimed at midcareer professionals who desire to work in data science and analytics but who have little or no background in programming. In particular, we describe how we use modern interactive computing platforms to accelerate the learning of our students both in and out of the classroom. We emphasize the importance of organizing the interaction with course material so that students learn not only to "think computationally" but also to "do computationally." We provide details of existing courses in computation offered at the Naval Postgraduate School, and we describe their ongoing evolution in response to increased demand from members of the civilian and military workforce.

History: This paper has been accepted for the *INFORMS Transactions on Education* Special Issue on The Education Science of Delivering Analytics Education.
Funding: This work was supported in part by funding from the Office of Naval Research.

Keywords: computation • data science • programming • end-user development

1. Why Teach Computation?

The explosive growth in demand for data science, analytics, machine learning, and artificial intelligence, both within the research community and among practitioners, is challenging traditional paradigms for education and training. Traditional disciplinary paths to the study and practice of these topics—that dive deeply into fundamentals of computer science, statistics, mathematics, and operations research—are frequently being eschewed in favor of hybrid programs that cover a mix of concepts and topics focused on solving immediate problems. Notwithstanding the ongoing debates regarding the best way to introduce students to the relevant theoretical underpinnings, there is a broad consensus on the need for students in these evolving disciplines to be proficient in *computation*.

There is a widely recognized need to teach *computational thinking* (CT) in support of science, technology, education, and/or mathematics education—see Wing (2006, 2008) and National Research Council (2010, 2011)—with considerable effort now dedicated to understanding how to integrate CT into K–12 educational programs (e.g., Barr and Stephenson 2011, Grover and Pea 2013, and Lye and Koh 2014). However, for organizations looking to hire skilled workers in data science and analytics, it is crucial that students are able to not only "think computationally" but also "do computationally" (Denning 2009). This recognition is consistent with the considerable gap between "learning" and "doing" in statistics (McNamara 2016).

Students in operations research (OR) at the Naval Postgraduate School (NPS) are midcareer, active duty military officers with unique needs for learning computation. First, most students have no prior background in programming or computation. Second, the applied nature of military OR and the short duration of our master's degree program mean that traditional approaches to teaching computer science are not appropriate or effective. Third, our students are unlikely ever to work as software developers, but many will use computation in their daily work and/or oversee contractors who build software systems. Fourth, our graduates will be tasked with advising senior leaders on issues related to the use of computation to support military operations, and many of them will become senior leaders themselves. As a result, it is imperative that our course offerings in computation are carefully designed and integrated within the rest of our curriculum to provide a rapid path to computational productivity.

This article provides an overview of how and why we take our students "from zero to hero" in their computational abilities over a short period of time. In particular, we highlight the use of modern interactive computing platforms, such as the Jupyter

Notebook, to achieve these results. We proceed in several steps. First, we describe the overall needs of our students and how these compare with broader needs of practitioners in military and nonmilitary settings. Next, we describe the way in which we use Jupyter notebooks, both inside and outside the classroom, to organize the way in which students engage computing concepts and materials. We also provide details of the organization and structure of these courses. Finally, we provide qualitative and quantitative evidence demonstrating the effectiveness of these courses in facilitating intended learning outcomes.

2. What to Teach in Computation?

There is no single standard defining the set of topics, concepts, and skills that are required for an individual to be proficient in computation. However, approximately 10 years ago, the INFORMS Computing Society (ICS) worked to identify key elements of an undergraduate curriculum aimed at the intersection of operations research and computer science (Hardin et al. 2012). With a retrospective look that considers the historical development of these fields going back to the 1960s, the ICS divides the relevant base of knowledge into three categories of skills:

• foundational skills: those essential for future work;

• core skills: those important for success and that are often associated with introductory graduate work;

• recommended skills: more-advanced skills useful to future work. (Hardin et al. 2012, p. 118)

Each of these categories is further subdivided into topics commonly associated with mathematics, computer science, and operations research. Within this trichotomy, the ICS also considers "the depth to which a subject should be known:

• competency/fluency: lucid control of the topic.

• experience: the central theme of a course.

• familiarity: a cursory introduction to the topic." (Hardin et al. 2012, p. 118)

Consideration for the depth of knowledge in these individual areas is of particular relevance for teaching and learning computation.

Denning (2002), borrowing from earlier works by Hubert Dreyfus and Fernando Flores, describes a "ladder of competence" for professional development that includes the following progression: "novice (beginner)," "advanced beginner (rookie)," "professional (competent)," "proficient professional (star)," "expert (virtuoso)," "master," and "legend" (p. 22). This framework is not novel (i.e., it clearly builds on the work of Hoffman (1998)), but it is noteworthy in that it includes both a general description of the abilities of an individual at each level and the modes of learning most often associated with it. The focus in Denning (2002) is ultimately on careers in information technology and software engineering; however, the "ladder" is provocative in that it focuses attention on what it means to be an "expert" in a given field and the progression one follows to get there.

It is important that our next generation of operations analysts and data scientists not only are able to execute well defined computational tasks but also can apply their knowledge to unfamiliar problems, draw connections to other experiences, evaluate novel situations, and ultimately create new knowledge for dealing with emergent challenges. This progression is consistent with classic theories of learning (e.g., Bloom et al. 1956) and more modern adaptations (e.g., Krathwohl 2002). And the ICS study is not alone in its attempt to identify the needs of students in computation (indeed, Hardin et al. (2012) cites a number of complementary efforts within other professional organizations).

3. How to Teach Computation?

Recent emphasis on the need for computational thinking in curricula has stimulated considerable discussion about how best to *teach* it, but there is long-standing recognition that this is not the same as what it takes to *learn* computation.

3.1. Teaching Is Not Learning

The phrase "teaching is not learning" comes from Doumont (2004, 2010), who recognizes that *teaching*, particularly in a traditional lecture, is an activity that involves an interaction between the instructor and the material, whereas *learning* is something that requires interaction between the student and the material. Doumont (2014) argues that the role of the instructor should not be about trying to "cover the material" but should instead be about "organizing the encounter" for the student to engage the material—specifically, to create an encounter in which to (1) allow the student *see* it, (2) allow the student *do* it, and (3) allow the student *discover* it.

3.2. Jupyter Notebooks

In the fields of scientific computing and computational sciences, there is a need for a *notebook*-like construct to serve as a means for documenting and reproducing computational workflow (Kluyver et al. 2016). The Jupyter Notebook—originally an outgrowth of the IPython project (Pérez and Granger 2007)—is a browser-based platform for embedding live computational workflow that can be saved and shared. As described in Kluyver et al. (2016, p. 88), "Notebooks are designed to support the workflow of scientific computing, from interactive exploration to publishing a detailed record of computation. The code in a notebook is organised into cells, chunks which can be individually modified and run. The output from each cell appears directly below it and is stored as part of the document." Although Jupyter notebooks were originally written solely to support the IPython interface (e.g., their initial name as "IPython notebooks" gives rise to their .ipynb file extension), Kluyver et al. (2016) note that there is now support for more than 100 backend kernels (Barba et al. 2019), including the popular R programming language (R Core Team 2019). In fact, the name for Project Jupyter was inspired in part from three of the most prominent scientific computing languages today: *Ju*lia, *Pyt*hon, and *R*, again emphasizing flexibility to support multiple computing environments (Pérez and Granger 2017).

Jupyter notebooks support a variety of rich text formats, making it easy to integrate formatted text and images in addition to live code. Specifically, individual cells can contain hypertext markup language (HTML), markdown (Gruber 2004), and LaTeX. Moreover, Jupyter notebooks can contain both static and dynamic visualizations generated from the code they execute. This makes Jupyter notebooks useful for a variety of tasks, ranging from exploratory learning and rapid prototyping to informal dashboards and user interfaces.

As of 2018, there were more than *two million* Jupyter notebooks on GitHub (Fenner 2018), with some of the best examples available from an online curated gallery (Project Jupyter 2020a). Jupyter has become a "de facto standard" not only for scientific computing but also for data exploration, classroom instruction, tutorials, and interactive software manuals (Perkel 2018). The IPython/Jupyter Notebook has even been categorized as one of "ten computer codes that transformed science" (Perkel 2021).

3.2.1. The Rise of "Computational Narratives." Not surprisingly, Jupyter notebooks are used for teaching traditional computer science (Zastre 2019), as well as a number of advanced topics, including high-performance computing (Glick and Mache 2018); Internet of things, cloud computing, and blockchain (Rao and Dave 2019); agile development (Guerra et al. 2019); and artificial intelligence (O'Hara et al. 2015). Moreover, Jupyter notebooks are commonly used for teaching computer science to non-CS students (Smith 2016, Vial and Negoita 2018).

Perhaps more important, however, "live code, narrative text, and visualizations are all being integrated together into documents that tell stories using code and data" (Pérez and Granger 2017). These *computational narratives* emphasize the importance of "programming for specific insight rather than generalization" (Pérez and Granger 2017).

3.2.2. Use Cases. It is relatively easy to find articles on the use of Jupyter notebooks for teaching concepts across a wide range of application areas, including physics (Urcelay-Olabarria et al. 2017), engineering (Cardoso et al. 2018), power systems dynamics and control (Milano and Jónsdóttir 2018), architecture (Maestre et al. 2016), optimization in manufacturing problems (Suárez et al. 2018), and industrial and systems engineering (Wang et al. 2017). Jupyter notebooks have even been used to develop written communication skills related to the organization and presentation of data and their analysis (Willis et al. 2020).

There also exist a wide variety of materials to support the adoption and use of Jupyter notebooks for instruction. These include articles to understand the basics for how to setup and use Jupyter notebooks in the classroom (Koehler and Kim 2018), challenges for developing and delivering courses in data science at scale (Swamy 2018), support tools such as nbgrader for automating the process of grading notebooks (Jupyter Project et al. 2019), and cloud computing for bootcamp-style short courses (Holdgraf et al. 2017).

When teaching nontraditional students (e.g., practitioners in industry), it is important to embed computation in a workflow that is designed with an analytic objective in mind, not simply programming for the sake of programming. For example, Kross and Guo (2019) interviewed "20 data scientists who teach in a diverse variety of settings across industry and academia" and found that a key motivating factor for their students is the desire to analyze their data; the study of computation is viewed solely as a means to an end and engaged somewhat grudgingly.

This need for *end-user programming*—defined by Ko et al. (2011, p. 4) as "programming to achieve the result of a program primarily for personal, rather [than] public use"—is common across domains of data science and analytics. It is particularly acute for military operations research analysts.

4. Computation at NPS-OR

Students at the Naval Postgraduate School are primarily midcareer military officers with approximately 7–10 years of operational experience beyond their undergraduate degree. Admission to the operations analysis curriculum requires past experience with mathematics (at the level of calculus), but typically, our students have no prior experience with computation or programming. For those students who have programmed in the past (e.g., in their undergraduate degree), we frequently hear that "it was a bad experience" or that they "didn't learn very much." For the most part, we are starting from scratch.

This lack of computational background for incoming students at NPS is in stark contrast to many graduate programs that often require some prior background in programming as a prerequisite for admission. In many graduate programs, students have the flexibility to choose the programming environment but are left on their own to apply it in the context of their coursework.

By contrast, we cannot assume this background. Indeed, a major strength of NPS is its ability to bring students with little or no prior background up to speed in a short period of time. The result is a master's degree program that runs approximately 24 months in length. Figure 1 shows the matrix of standard courses taken by students who pursue the "data analytics track" for the Master of Science in Operations Research degree. NPS operates on four 12-week academic terms (called "quarters") per year. The program starts with a "refresher quarter" targeted at students who have been away from school more than five years (the majority of our students). Every student is required to complete a master's thesis. Depending on where they are in their career progression, some students are also required to complete an additional four-course sequence in joint professional military education (JPME) through the Naval War College, which adds an additional academic term to their residence.

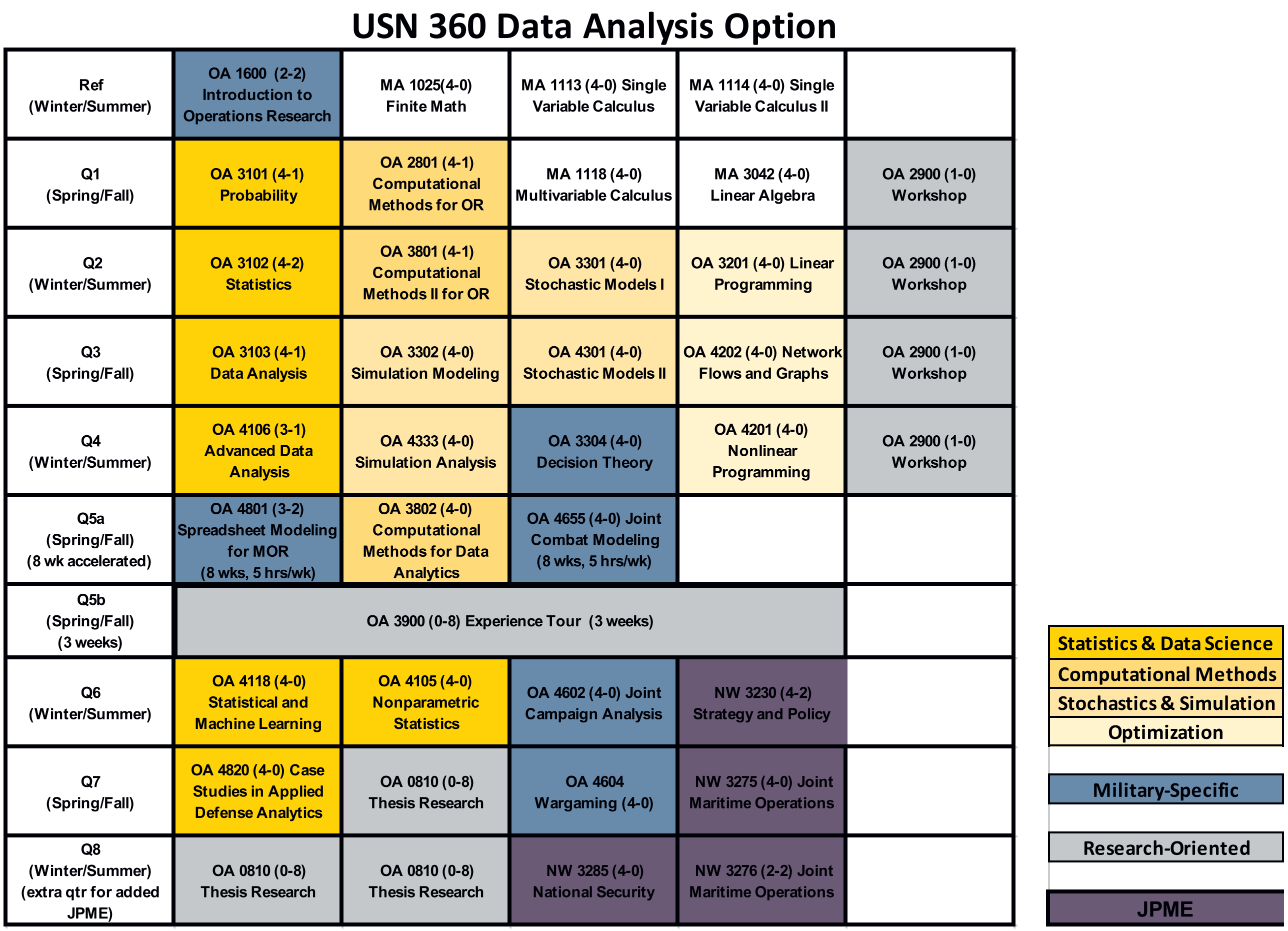
All students in OR degree programs at NPS are currently required to take a two-quarter sequence in introductory computation. Students in the data analytics track take additional advanced courses. We describe the goals, organization, and methods of each.

4.1. Computational Methods for OR, Part I

Students in the first nonrefresher quarter (called "Q1") take OA2801: Computational Methods for Operations Research. This course meets five days per week for 50 minutes per class—four lectures and one laboratory session per week—over a period of 11 weeks.

The course is organized around a weekly cycle: lectures Monday through Thursday introduce new material that builds toward a laboratory assignment, which is introduced and started during Friday's session. The laboratory assignment represents a practical computational

Figure 1. Students Following the "Data Analytics" Track in the Operations Analysis Curriculum at the Naval Postgraduate School Take Four Courses per 12-Week Term over a Period of Two Years



USN 360 Data Analysis Option

Ref (Winter/Summer)	OA 1600 (2-2) Introduction to Operations Research	MA 1025(4-0) Finite Math	MA 1113 (4-0) Single Variable Calculus	MA 1114 (4-0) Single Variable Calculus II	
Q1 (Spring/Fall)	OA 3101 (4-1) Probability	OA 2801 (4-1) Computational Methods for OR	MA 1118 (4-0) Multivariable Calculus	MA 3042 (4-0) Linear Algebra	OA 2900 (1-0) Workshop
Q2 (Winter/Summer)	OA 3102 (4-2) Statistics	OA 3801 (4-1) Computational Methods II for OR	OA 3301 (4-0) Stochastic Models I	OA 3201 (4-0) Linear Programming	OA 2900 (1-0) Workshop
Q3 (Spring/Fall)	OA 3103 (4-1) Data Analysis	OA 3302 (4-0) Simulation Modeling	OA 4301 (4-0) Stochastic Models II	OA 4202 (4-0) Network Flows and Graphs	OA 2900 (1-0) Workshop
Q4 (Winter/Summer)	OA 4106 (3-1) Advanced Data Analysis	OA 4333 (4-0) Simulation Analysis	OA 3304 (4-0) Decision Theory	OA 4201 (4-0) Nonlinear Programming	OA 2900 (1-0) Workshop
Q5a (Spring/Fall) (8 wk accelerated)	OA 4801 (3-2) Spreadsheet Modeling for MOR (8 wks, 5 hrs/wk)	OA 3802 (4-0) Computational Methods for Data Analytics	OA 4655 (4-0) Joint Combat Modeling (8 wks, 5 hrs/wk)		
Q5b (Spring/Fall) (3 weeks)	OA 3900 (0-8) Experience Tour (3 weeks)				
Q6 (Winter/Summer)	OA 4118 (4-0) Statistical and Machine Learning	OA 4105 (4-0) Nonparametric Statistics	OA 4602 (4-0) Joint Campaign Analysis	NW 3230 (4-2) Strategy and Policy	
Q7 (Spring/Fall)	OA 4820 (4-0) Case Studies in Applied Defense Analytics	OA 0810 (0-8) Thesis Research	OA 4604 Wargaming (4-0)	NW 3275 (4-0) Joint Maritime Operations	
Q8 (Winter/Summer) (extra qtr for added JPME)	OA 0810 (0-8) Thesis Research	OA 0810 (0-8) Thesis Research	NW 3285 (4-0) National Security	NW 3276 (2-2) Joint Maritime Operations	

Statistics & Data Science
Computational Methods
Stochastics & Simulation
Optimization
Military-Specific
Research-Oriented
JPME

Notes. Courses in computation are foundational to work in the "pillars" of optimization, statistics, stochastic processes, and simulation. In turn, these pillars support advanced course work in warfare analysis and data analytics. Every student also completes a master's thesis.

exercise that students complete on their own and submit for grading the following Wednesday evening. There are a total of seven laboratory assignments during the 10-week term. The course includes a weekly pencil-and-paper quiz (typically the first 10 minutes of Thursday's class) that is graded but mostly intended to provide quick feedback on a student's conceptual understanding of the material. The course has a midterm examination (typically around week 6) and a final examination (after week 10), both of which are traditional pencil-and-paper exercises.

Because we cannot be certain which computational environment(s) our student-officers will have in their work after graduation, we believe it is important that they have familiarity with a general-purpose programming language. We have used the Python programming language (Python Software Foundation 2020) in this course since 2012, although previously, we had been using the Java programming language. As of 2014, Python has become the most popular language for teaching introductory programming (Guo 2014).

The course integrates live computation in its sessions on a daily basis. Students are required to bring their laptops to class every day, and after some initial setup during the first week, each lecture is based on a daily Jupyter notebook that the instructor and student execute together during the class session. We require students to use the same Python installation so that we maintain a consistency in their experience.

We emphasize from the outset that this is not a programming course in Python but rather a course on solving problems using computation. That is, the emphasis is on having students learn how to implement and analyze basic algorithms relevant to operations analysis. To do this effectively, a student needs to understand fundamental trade-offs in representing and manipulating data on a computer and how these affect the outcome of any numerical analysis. This includes both the correctness and efficiency of algorithms. The goal is to elevate the conversation above the syntax of "how to do it in Python," as most concepts are independent of the language, and also because in order to "get the job done," an analyst will sometimes need to open up a book and learn a new computing language and/or environment. Moreover, as an analyst, a graduate of our curriculum will need to be able to use the computer to produce results that (hopefully) provide insight into decisions. Ultimately, it is the analyst (not the computer) who is responsible for any recommendations that are made based on those results. Thus, a critical skill for the student is to discern if and when to believe the results that the computer produces. In other words, the challenge posed to the student is not merely to get the program to produce output but to answer the fundamental question: How do you know if the program "works"?

Figure 2 shows the syllabus for the course as it was last taught in the first quarter (Q1) of academic year 2021 (the fall of calendar year 2020). The laboratory assignments drive the overall pace of instruction and are carefully selected to integrate into other coursework. The first assignment starts with the use of computer as a calculator and involves computing great circle distances and navigation (spherical trigonometry). The next labs involve basic numerical recipes for calculus—specifically, numerical integration and root finding—and a means for working with loops and learning about iterative algorithms. By week 4, students learn how to open a text file and parse its data, which enables them to work with real data in comma-separated value format. When combined with the use of basic list and dictionary data structures, this also paves the way to more complicated problems and solution techniques. Each laboratory assignment involves the use of real data sets, typically thousands of lines each, and these laboratory assignments not only require them to develop new skills in computation but also reinforce the idea of computation in support of some broader analytic question. The progress is dramatic; in the final laboratory assignment, the students implement a recursive stack-based algorithm to enumerate all possible paths for a multihop aircraft routing problem.

The ambitious schedule in this course is made possible only through the daily use of Jupyter notebooks, both in class and outside of class. In class, the Jupyter notebook provides a "script" for the computational narrative that is the focus of each session. Students follow along with the instructor, executing the code together one cell at a time, and the incremental read-evaluate-print loop nature of the interaction—the basis of interactive programming (see Iverson (1962) and Spence (1975))— allows the material to build incrementally. In the spirit of Doumont (2014), the instructor can use a well-designed notebook to allow the student not only to *see it* and to *do it* but also to *discover* key concepts in computation. And because notebook cells can be added, edited, or rerun on the fly, there is unlimited opportunity for improvisation. Student questions can be addressed with a "let's try it together" excursion that reinforces the power of experimentation as a form of self-learning. Thus, instead of an instructor "lecturing at the students," the classroom encounter has been organized such that the instructor and students are engaging and experiencing the material together.

The use of "live computation" in the classroom is not novel; discussions about the use of computers in the classroom go back (at least) to the 1980s (Papert 1990). However, the use of a computer for a live demonstration is challenged by the need for students to (i) follow along with the syntax prescribed by the instructor and (ii) correctly type the appropriate syntax and

Figure 2. OA2801: Computational Methods for Operations Research, as It Was Taught at the Naval Postgraduate School in Q1 AY2021

Week	Lessons				Labs
1	**28-Sep** 0. Course Introduction	**29-Sep** 1. Intro to Computation & Programming	**30-Sep** 2. Representing Data (Data Types)	**1-Oct** 3. Numerical Error	**2-Oct** Lab 0. Python Setup
2	**5-Oct** 4. Expressions (Math & String Operators) HW 1 Due	**6-Oct** 5. Expressions (Logical & Boolean Operators)	**7-Oct** 6. Functions (Intro) HW 2 Due	**8-Oct** 7. Conditionals Quiz 1 HW 3 Due	**9-Oct** Lab 1. Great Circle Distance
3	**12-Oct** **Columbus Day (No Class)**	**13-Oct** 8. While Loops	**14-Oct** 9. Nested Loops Lab 1 Due	**15-Oct** 10. Conditionals Quiz 2	**16-Oct** Lab 2. Estimating Integrals
4	**19-Oct** 11. Functions (Part 2)	**20-Oct** 11. Namespace	**21-Oct** 13. Printing & Debugging Lab 2 Due	**22-Oct** 14. Iterative Algorithms Quiz 3	**23-Oct** Lab 3. Finding Roots
5	**26-Oct** 15. Strings	**27-Oct** 16. Tuples	**28-Oct** 17. Lists (Part 1) Lab 3 Due	**29-Oct** 18. Lists (Part 2) Quiz 4	**30-Oct** 19. Lists (Part 3)
6	**2-Nov** 20. Multi-Dimensional Lists	**3-Nov** 21. Parsing Text & File I/O	**4-Nov** 22. Batch Mode	**5-Nov** 23. Command Line Arguments Quiz 5	**6-Nov** Lab 4. Data Management (File I/O & Parsing)
7	**9-Nov** 24. Basic Plotting	**10-Nov** 25. Review	**11-Nov** **Veteran's Day (No Class)**	**12-Nov** Midterm Exam	**13-Nov** Midterm Solution Review
8	**16-Nov** 26. Dictionary (Part 1)	**17-Nov** 27. Dictionary (Part 2)	**18-Nov** 28. Sets & Sorting Lab 4 Due	**19-Nov** 29. Sorting	**20-Nov** Lab 5. UAV Data (Selection & Sorting)
9	**23-Nov** 30. Function Defaults & Redirecting I/O	**24-Nov** 31. IDEs, Debugging	**25-Nov** No Class Lab 5 Due	**26-Nov** **Thanksgiving Holiday**	**27-Nov** **Thanksgiving Holiday**
10	**30-Nov** 32. Intro to Comp Complexity	**1-Dec** 33. Big O Analysis	**2-Dec** 34. Searching	**3-Dec** 35. Sorting & Complexity Quiz 6	**4-Dec** Lab 6. Eastern Airports
11	**7-Dec** 36. Random Number Generation	**10-Dec** 37. OR Application: Monte Carlo Methods	**11-Dec** 38. OR Application: Craps Overivew Lab 6 Due	**12-Dec** 39. OR Application: Craps Exercise Quiz 7	**13-Dec** Course Wrap Up & Final Exam Review
12	**14-Dec**	**15-Dec**	**16-Dec**	**17-Dec** FINAL EXAM	

commands. The Jupyter notebook removes the need for students to have to copy down what the instructor is saying or typing because they already have the script to follow along. However, the script is flexible enough so that as questions arise, the instructor and students can explore new ideas. This encourages students to "try things" both during and outside of class, and it creates a platform for self-discovery that is a natural outgrowth of the classroom experience. In many cases, students might not know how to discover on their own. But by doing it together in class, the instructor reinforces the idea that "doing computation is not a big deal" and instead is something that the students can (and need to) practice daily, both in and out of class.

These Jupyter notebooks also play an important role outside of class. Because these notebooks are made available online in advance of each class (this is important), students can prepare for each session by exploring these narratives on their own, which helps them to identify in advance questions that need attention during class. If the instructor improvises during the class in response to questions or discussion, his or her Jupyter notebook also serves as a record of what was done and can be posted and made available to the class. Thus, the notebook becomes an important tool for reviewing material after class, for study, or as a reference to be revisited in the future.

Just as any other teaching device, Jupyter notebooks on their own do not automatically create a positive classroom experience. An instructor that does nothing more than read aloud what is in the notebook will quickly have the students rethinking whether there is a need to attend class. Instead, the value of attending class comes from the ability to pivot the discussion in response to student engagement and questions, in a manner that students can actually follow.

This first course has a few other features worth noting. For each laboratory assignment, students are placed in *study groups* of (typically three) students. Students are allowed to collaborate freely with members of their study group (i.e., they can look at each other's code and help with debugging); however, at no point are they allowed to possess another student's code, either electronically or in printed form (i.e., each student has to type in their own code in his or her own "voice"). Study groups are not permitted to collaborate; if a group gets stuck, they can always seek help from the instructor. In order to minimize groupthink and maximize the potential for learning from different approaches and styles, these study groups are assigned (at random) by the instructor and change with each laboratory assignment. Although students can collaborate on their laboratory assignments, each student is required to participate in one-on-one *code review* meetings with the instructor at several points during the course. These short (e.g., 15-minute) meetings are focused on the most recent laboratory assignment, and they typically start with the instructor asking, "How did it go? Tell me what you did. … " These conversations are intended to be low stress but are incredibly revealing—within a few minutes, it becomes obvious how comfortable the student is with the concepts and practice of computation. By reviewing the assignment together, the instructor can identify student misconceptions and provide recommendations for improving the quality of the written code. It also provides an opportunity for the student to ask specific questions that do not arise in class. These code reviews are a large time investment—particularly given that there are no teaching assistants at NPS, so the instructor is doing all the lecturing, grading, and office hours, among other things—but this investment always pays dividends in student learning outcomes, and it typically also makes it easier to grade the laboratory assignment itself.

At the end of this first course, students are proficient in basic computation in Python using Jupyter notebooks. However, Jupyter notebooks are not ideal for all computational tasks. In particular, a complicated workflow that requires automation should be implemented in the form of a traditional stand-alone program or script that can be executed from the command line terminal or within an integrated development environment (IDE). Long blocks of code should be organized in functions that can be debugged once and then used when needed. All the principles of good software engineering still apply, and students are slowly introduced and taught best practices for organizing and managing their code.

In parallel to this course, the students also use Python in an introductory probability course (see Figure 1) to perform computation on applied problems. In the future, we hope to similarly integrate the use of computation into the linear algebra course that is also taken concurrently.

4.2. Computational Methods for OR, Part II

During the second quarter (Q2), students take the sequel course, OA3801: Computational Methods for Operations Research II. This course is also organized around 50-minute class sessions, but it meets only four days per week. Rather than being driven by a weekly laboratory assignment, the course is divided into modules that delineate key concepts. Laboratory assignments are used as practical exercises, but there are fewer of them. The final exam is replaced with a final project; however, there are two midterm examinations that are administered as *live coding exams*.

The exams in this course follow a two-part format and are implemented using Jupyer notebooks, as follows. Part 1 of the exam begins at the start of the class session, whereby students are given access (via the course website) to a Jupyter notebook and corresponding data file. The notebook contains instructions and questions for the exam in the form of a computational narrative. That is, the exam is focused on a particular data set, and the exam questions require the student to perform a sequence of computations on the data. Students answer the questions in the cells of the notebook and submit the notebook on the course website at the end of the hour. This is followed by a second, take-home portion of the exam. Upon submission of the first notebook, the students are given access to a second Jupyter notebook that they must complete outside of class. This second notebook contains all of the same instructions and questions as the first notebook, but it typically includes a few more questions that extend the narrative and analysis. Students submit the second notebook via the course website, due by the start of class the next day. During both the in-class and take-home portions of the exam, students are given free access to course materials and online help. The only restriction is that they are not allowed to interact with another person either inside or outside the class. The two notebooks are graded independently for an overall combined score.

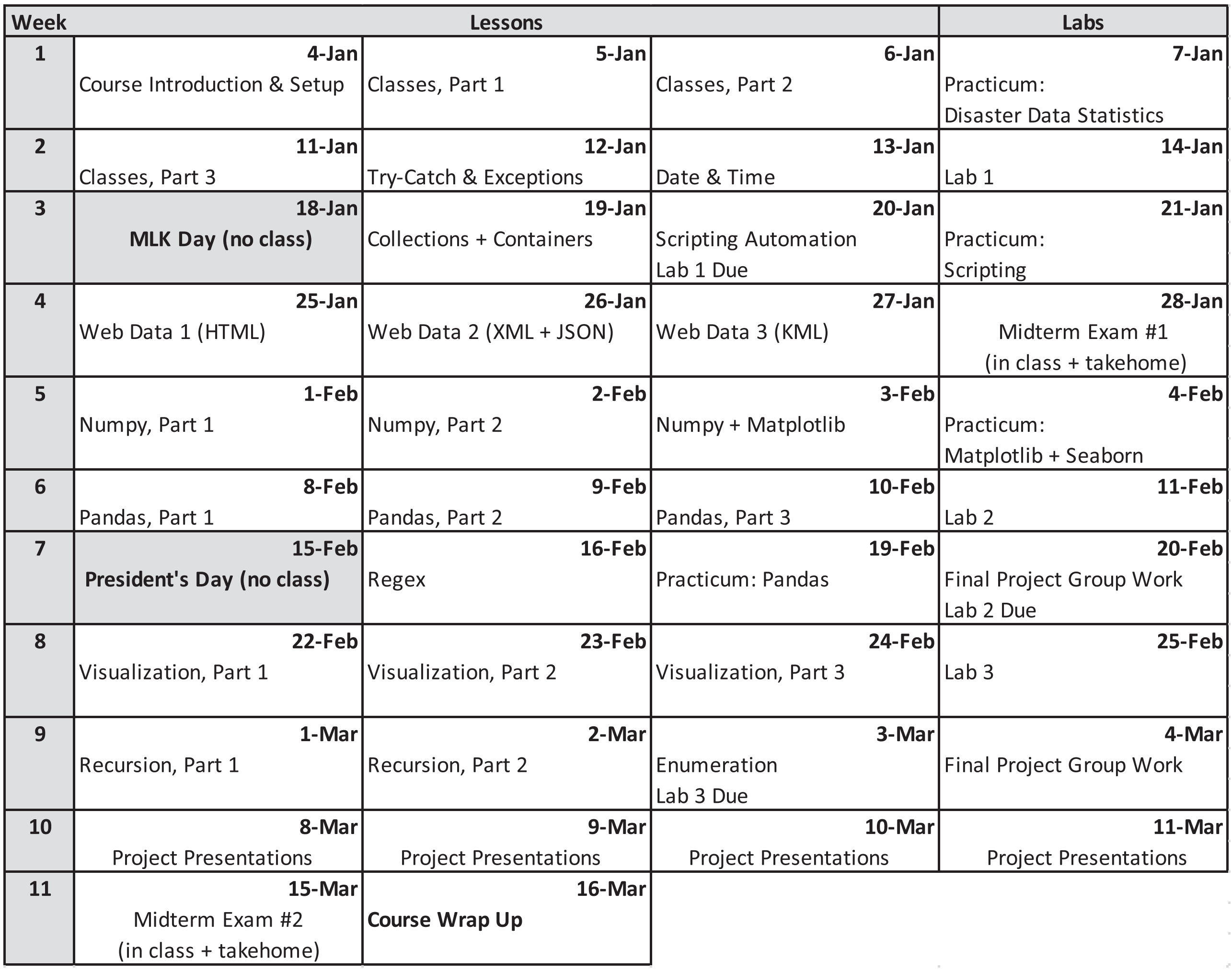
The in-class portion of the exam is time constrained to 50 minutes, and accordingly, it measures the ability of the student to quickly conceive and engage the computational task presented. Relatively speaking, the take-home portion of the exam is not time constrained, and it is intended to assess how the student can perform on his or her own, given ample time. On the take-home portion, students can (and typically do) submit responses that differ from what they submitted during the in-class portion.

These live-coding exams create a learning experience as well as an opportunity for assessment. Many students who were comfortable in OA2801 working on weeklong laboratory assignments in the comfort of a study group are less comfortable or less capable when having to work on their own in a timed environment. However, history shows that students learn a tremendous amount from this first experience and adapt for the second exam. Most students who struggle on the first midterm exam show dramatic improvement and often excel on the second midterm exam.

In terms of its content, this second course continues to focus on representation and analysis of data using algorithms and data structures; however, it additionally introduces the basics of object-oriented programming and software design, and it focuses on student proficiency for manipulating and visualizing data. Figure 3 shows the syllabus for the course as it is taught in Q2 of academic year 2021 (the winter of calendar year 2021). The first module of the course focuses on Python classes and how they work. The goal is for students to understand how to create their own "customized containers" for data when needed, as well as to understand more deeply how Python works. The second module introduces the NumPy library (Oliphant 2006) and the notion of vectorized computation, with considerable time dedicated to introducing and practicing the use of pandas (McKinney 2010) for large-scale data analysis. The third module is dedicated to collecting and manipulating data commonly found on the internet, including web scraping of HTML pages, eXtensible Markup Language (XML), and Javascript Object Notation (JSON). The fourth module exposes students to the rapidly expanding suite of Python-based visualization tools now freely available.

In lieu of a final examination, this course has a final project. Students self-select into teams (approximately three students each) and execute a project of their own choosing. The requirements are that the project involves (1) using a real data set obtained on the internet;

Figure 3. OA3801: Computational Methods for Operations Research II, as It Was Taught at the Naval Postgraduate School in Q2 AY2021



Week	Lessons			Labs
1	**4-Jan** Course Introduction & Setup	**5-Jan** Classes, Part 1	**6-Jan** Classes, Part 2	**7-Jan** Practicum: Disaster Data Statistics
2	**11-Jan** Classes, Part 3	**12-Jan** Try-Catch & Exceptions	**13-Jan** Date & Time	**14-Jan** Lab 1
3	**18-Jan** **MLK Day (no class)**	**19-Jan** Collections + Containers	**20-Jan** Scripting Automation Lab 1 Due	**21-Jan** Practicum: Scripting
4	**25-Jan** Web Data 1 (HTML)	**26-Jan** Web Data 2 (XML + JSON)	**27-Jan** Web Data 3 (KML)	**28-Jan** Midterm Exam #1 (in class + takehome)
5	**1-Feb** Numpy, Part 1	**2-Feb** Numpy, Part 2	**3-Feb** Numpy + Matplotlib	**4-Feb** Practicum: Matplotlib + Seaborn
6	**8-Feb** Pandas, Part 1	**9-Feb** Pandas, Part 2	**10-Feb** Pandas, Part 3	**11-Feb** Lab 2
7	**15-Feb** **President's Day (no class)**	**16-Feb** Regex	**19-Feb** Practicum: Pandas	**20-Feb** Final Project Group Work Lab 2 Due
8	**22-Feb** Visualization, Part 1	**23-Feb** Visualization, Part 2	**24-Feb** Visualization, Part 3	**25-Feb** Lab 3
9	**1-Mar** Recursion, Part 1	**2-Mar** Recursion, Part 2	**3-Mar** Enumeration Lab 3 Due	**4-Mar** Final Project Group Work
10	**8-Mar** Project Presentations	**9-Mar** Project Presentations	**10-Mar** Project Presentations	**11-Mar** Project Presentations
11	**15-Mar** Midterm Exam #2 (in class + takehome)	**16-Mar** **Course Wrap Up**		

(2) using the tools from the course to represent, manipulate, and analyze the data; and (3) visualizing the results in a manner that provides insight into a question of interest. Student teams present their project in class during the last week of the term, and they submit a short executive summary of their findings along with their source code and data. The objective is to integrate their computational thinking and doing into a practical exercise that can be shared with the class and whereby everyone learns from the collective experience.

In parallel to this course, the students are involved in two other courses making significant use of computation (again, see Figure 1). In OA3201: Linear Programming, students use the Python-based pyomo module (Hart et al. 2011, 2017) to formulate and solve mathematical programs. In OA3102: Statistics, students are introduced to the R programming language (R Core Team 2019), which is used throughout the statistics sequence of courses. In general, students learn both Python and R in the NPS-OR curricula. There is some support and coordination for the introduction and use of R in OA3801; in particular, Jupyter notebooks can also be used with R and provide the same opportunity for accelerated learning as with Python.

4.3. Advanced Courses in Data Analytics

Computation pervades all courses in the NPS-OR curriculum. However, students in the data analytics track take two specific courses in advanced computation.

The course OA3802: Computational Methods for Data Analytics is also module and project based, with a few significant differences. There is an emphasis on working with very large data sets both on customized high-performance computing systems and in cloud-based environments (e.g., Amazon Web Services). The course uses both Python and R to support computation, with an emphasis on seeing the pros and cons of each for individual tasks. In general, students get to choose which of these computational environments they use for each assignment. Moreover, student work requires more self-study and discovery to work through challenges common to data in the real world.

As a final capstone, students pursuing the data analytics track also take the course OA4820: Case Studies in Applied Defense Analytics, in which they are organized into teams and work with a real sponsor on a real problem using real data over the span of an 11-week period. They iterate with the sponsor on a weekly basis, rapidly prototyping a specific analytic tool in the spirit of lean software development (e.g., Poppendieck and Poppendieck 2003), ultimately delivering a functional tool by the end of the term.

4.4. Short Courses in Computation

There is a real and ongoing need among practitioners to refresh, update, or expand their computational skills. Many of the computational tools and libraries that are the mainstay of data analytics did not exist as little as 5 or 10 years ago. Although there are now a large number of online courses, both for fee and free, it is often unclear which of these will be most beneficial. As a result, there is a rise in demand for short bootcamp-style courses in computation. Within the last 18 months, the author has delivered short courses in computation for the U.S. Naval Academy, U.S. Naval Warfare Centers (laboratories), and the U.S. Marine Corps, as well as tutorial sessions at conferences sponsored by both INFORMS and the Military Operations Research Society (MORS). Although short-course instruction is not a primary mission for NPS, there is growing demand for this type of career-long learning within both the civilian and military workforces.

These short courses have all been delivered using Jupyter notebooks, and all the advantages of notebooks for accelerated learning in traditional residential courses hold (and are even more important) in these limited encounters. However, there is one key difference in the delivery of these short courses. It often takes a few hours to get a room of students set up with a local Python installation (assuming that all students have administrator privileges), and this is typically too much time to lose in a short course. Rather, we often use a cloud-based computing environment such as CoCalc (a paid service designed for instruction; see https://cocalc.com) or Google Colaboratory (a free service; see https://colab.research.google.com). These services allow the user to run a Jupyter notebook in a web browser, but instead of running the notebook on the local machine, the notebook is hosted on a server in the cloud.

4.5. Instructional Changes During COVID-19

Along with the rest of the world, NPS moved to strictly online instruction in March 2019 to limit the spread of the novel coronavirus. As a school, NPS is subject to restrictions imposed by the Department of the Navy and the State of California. Over the last year, there have been two complete iterations of the introductory OA2801–OA3801 sequence. For the most part, these courses did not need to change substantially. Because our use of Jupyter notebooks already had students following along on their own computers, the transition to an online interactive format (via Zoom) was relatively straightforward. Lectures remained largely the same and were recorded for those who were unable to attend or wanted to rewatch a lesson. Exams were administered via the course website, and they made use of Jupyter notebooks. Student meetings and presentations occurred online.

The most significant impact resulting from the online format was the reduction in (or lack of) direct student interaction. The usual opportunities for in-residence students to interact informally in the few minutes between classes, walking across campus, at lunch, or in the library largely disappeared. Students have access to videoconferencing or social media platforms, but synchronous interaction often requires explicit coordination and possibly a scheduled appointment. In a time of "too many Zoom calls," much of this interaction simply stopped. Moreover, the majority of our students fall into one of two categories: either they have young families that require their attention at home or they are young and single and often living alone. Both situations create additional challenges for long-term remote learning.

Overall, this lack of interaction has hurt the weaker students disproportionately. Whereas the stronger student can often manage the academic burden of increased self-study, the weaker student often struggles. In particular, our faculty did not fully appreciate the extent to which the weaker students are supported by a variety of mechanisms associated with being on campus. Indeed, the ethos of "leaving no person behind" is strong within our student body, but the lack of direct interaction limits the extent to which the group can support itself.

One significant change in the introductory OA2801 course was to replace one lecture a week with a "practicum" session devoted to student team work. Specifically, on Thursday each week, the students are divided into study groups and given the task of navigating a lesson together via a Jupyter notebook filled with code examples intended to stimulate self-discovery and practice. Students are required to submit the practice problems in order to receive a solution set, but these problems are not graded. The study groups remain the same for the practicum and laboratory sessions that week, giving the students a bit more time together. Initial feedback suggests that this is a welcome change and one that is likely to be continued when in-residence instruction resumes.

5. Assessment

An understanding of *what* and *how* to teach when it comes to computational thinking and doing is only one part of the puzzle. An ongoing challenge is to understand "how will we know if we have succeeded?" (e.g., Brown 2004).

Measuring success in the delivery of education for computational thinking has unresolved issues. Zhong et al. (2016) provide recent reviews on the challenges associated with assessing the effectiveness and outcomes for teaching computational thinking in K–12 education. As noted by Denning (2017), part of this challenge is because ongoing efforts to educate students in CT still suffer from vague definitions, a lack of effective assessment techniques, and overreaching claims that everyone needs to learn CT. More pragmatically, he argues that our focus should remain on "making educational offers that help people learn to be more effective in their own domains through computation" (Denning 2017, p. 39). That is precisely what we are doing—making military operations research analysts more effective at addressing real-world problems through the use of computation.

Since 2012, there have been a total of 593 students who have taken OA2801 and 270 students who have taken OA3801. On the surface, the success of these courses is evident in the increased computational proficiency of the students in their downstream courses and thesis research; as a whole, they go "farther and faster" in computation than at any point previously.

In this section, we present some quantitative evidence of improvement in learning outcomes during the author's tenure, a period that spans approximately 15 years. The evidence comes in three forms: (1) student course evaluations required by NPS at the end of every term, (2) a direct survey of the most recent cohort of students to take these courses, and (3) unsolicited feedback from senior students and recent graduates.

5.1. Student Course Evaluations

The computation courses at NPS-OR undergo continuous change (and hopefully, improvement) as part of the routine business of graduate instruction. Course evaluations are required from every student for every course, and the results are stored and available for future use. Course evaluations are standardized—the same questions are asked for every course—however, there was a change in these questions in 2018. Because graduate instruction is a primary mission for NPS, teaching evaluations and student feedback are taken seriously and considered explicitly in promotion and tenure processes.

For analysis, the author reviewed student evaluations for 24 courses taught between 2007 and 2021; by coincidence, these were split 50/50 across OA2801 and OA3801. Table 1 shows five representative questions from course evaluations that were specifically considered in this analysis. The intent in selecting these questions was to understand the use of time in the classroom and effectiveness in allowing students to engage the material.

The responses to these questions follow typical sentiment analysis and use a Likert scale with rank order (Likert 1932). Specifically, students can select one of the following: (1) strongly disagree, (2) disagree, (3) neutral/no opinion, (4) agree, or (5) strongly agree. Figure 4 shows the results of these student surveys for the questions in Table 1 during three discrete periods. We use a diverging stacked bar chart, which is commonly believed to be most useful for this type of data. Figure 4(a) and 4(b) shows the responses to the

Table 1. Five Relevant Questions on NPS Course Evaluations Prior to (Left) and Since (Right) January 1, 2018

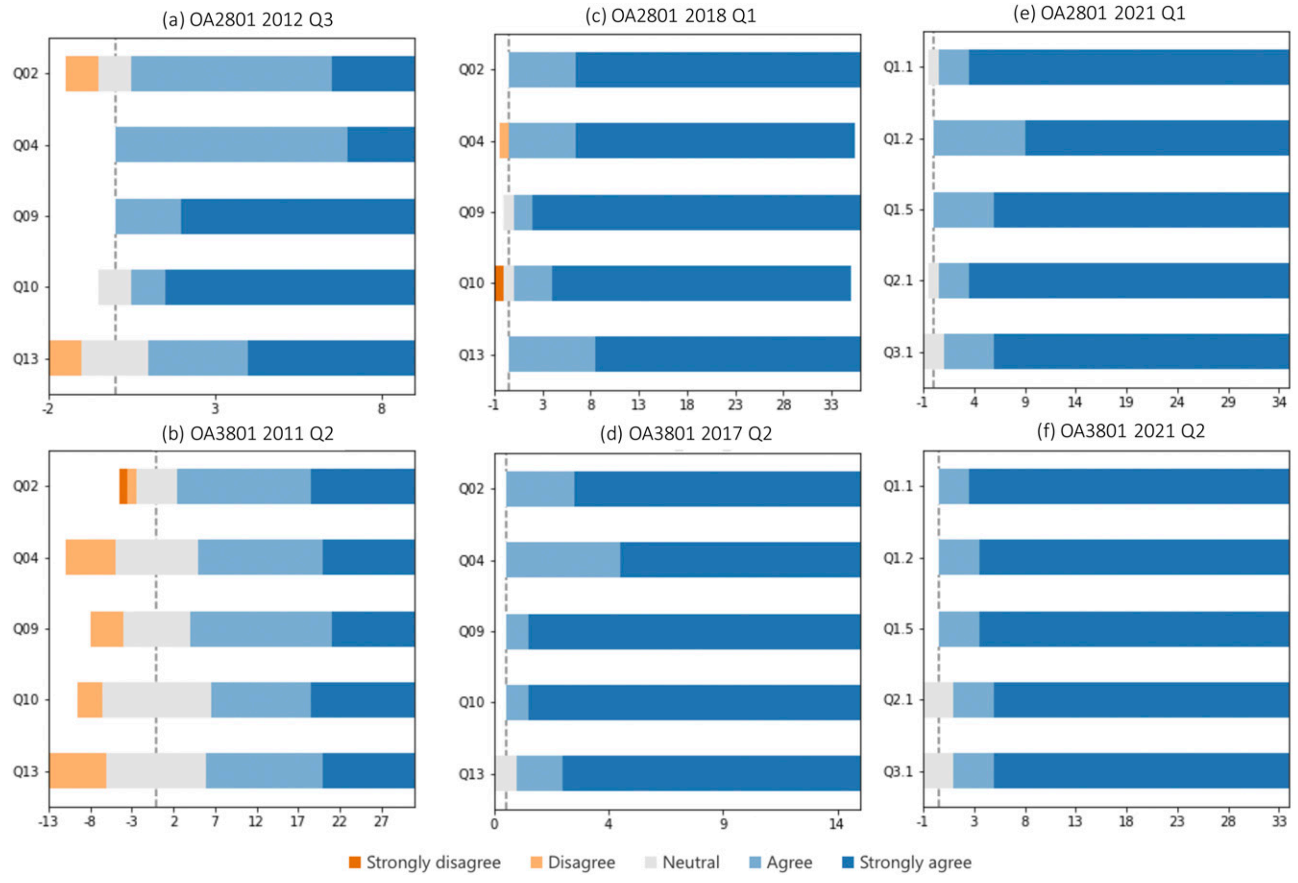
NPS student opinion form questions (prior to Jan 1, 2018)	NPS course evaluation form questions (since Jan 1, 2018)
Q2. Time in class was spent effectively.	Q1.1. I developed new skills and abilities.
Q4. Difficult concepts were made understandable.	Q1.2. I improved my understanding of the subject.
Q9. The instructor made the course a worthwhile learning experience.	Q1.5. Overall, I learned a great deal.
Q10. The instructor stimulated my interest in the subject area.	Q2.1. The course material engaged me in the subject matter.
Q13. Overall, I would rate this course: …	Q3.1. The instructor created a productive classroom environment.

questions in Table 1 (left) during the period 2011–2012, when the Python programming language was first introduced into these courses. Figure 4(c) and 4(d), shows the responses to the questions in Table 1 (left) during the period 2017–2018, after the courses had been redesigned around the use of Jupyter notebooks. Figure 4(e) and 4(f), shows the responses to the questions in Table 1 (right) as they were last taught this year, as in Figures 2 and 3, respectively.

Overall, the results in Figure 4 show an improvement in course outcomes as recorded in student course evaluations. The number of responses that select "disagree" or "strongly disagree" are reduced, and the number of responses for "strongly agree" are increased. In the most recent instantiation of these courses (Figure 4(e) and 4(f)), nearly all respondents selected "agree" or "strongly agree" for these questions.

The results in Figure 4 need to be viewed with a healthy dose of caution. No two iterations of a course were ever the same; the material covered and problems changed over time, as did the order and depth of some lessons. Additionally, the background and ability of the students is not the same from one cohort to the next. Even when we have two subsequent cohorts of similar ability, over time, we tend to ask them to do more in their coursework and research. Moreover, the

Figure 4. Progression of Student Course Evaluations for Computation Courses over the Last Decade



Notes. Raw counts of responses to questions from Table 1 demonstrate some improvement over three time periods. Panels (a) and (b) show results during the period when the Python programming language was first introduced into these courses. Panels (c) and (d) show results after the courses had been redesigned around the use of Jupyter notebooks. Panels (e) and (f) show the results for the most recent delivery of these courses. Figures created in Python are based on code developed by Bart (2018).

questions in the two columns of Table 1 (showing evaluations before 2018 and from 2018 onward) do not directly correspond one to one with each other, making most recent assessments not directly comparable with those in the past. In summary, although we see qualitative evidence of improvement for course outcomes that coincides with our use of the Jupyter notebook (among other things), the ongoing evolution of these courses was not designed as a scientific experiment, and therefore the extent to which we can statistically prove significance (e.g., with a *t*-test) is limited.

5.2. Survey of Recent Students

As a second form of evaluation, we surveyed students who most recently completed the OA2801 and OA3801 sequence of courses this year. A total of 39 students completed these courses; however, only 27 of them completed a voluntary (and anonymous) survey intended to serve as feedback to improve the experience and outcomes for future students.

Figure 5 shows responses to questions about the self-assessed ability of students to perform computational tasks both before and after taking OA2801. The computational tasks in the questions are taken directly from the stated learning objectives for the course. The dramatic improvement in responses serves as evidence of the effectiveness of the overall course design and method of instruction.

Figure 6 shows responses to questions about the self-assessed ability of students to perform computational tasks after taking OA3801 (the end of the introductory computation sequence). These questions are targeted at the confidence and ability of students to both "think computationally" and "do computationally" in their downstream courses and thesis. Again, the overall responses demonstrate successful outcomes for this sequence of instruction.

In addition, we asked these students the following question: "Looking across both OA2801 and OA3801, how important were [individually named] elements to helping me learn?" with possible responses of (1) distracting, (2) not useful, (3) neutral, (4) useful, and (5) essential. The top five responses among the 27 completed surveys were the following:

- Jupyter notebooks: essential (96.3%) and useful (3.7%)
- Direct instructor engagement: essential (77.8%) and useful (22.2%)
- Laboratory assignments: essential (77.8%), useful (14.8%), neutral (0%), not useful (3.7%), and distracting (3.7%)
- Organization of the course website: essential (70.4%), useful (25.9%), and neutral (3.7%)
- Official online Python documentation: essential (66.7%), useful (25.9%), neutral (3.7%), and not useful (3.7%)

It is clear that Jupyter notebooks are an important component of this learning progression.

5.3. Anecdotal Evidence from the Force

The final source of evidence is purely anecdotal and comes from unsolicited responses or correspondence that instructors receive from current or past students. Much of this feedback tends to be verbally communicated, but sometimes, students will take the time to write their thoughts, which is always appreciated. One such quote was recently received from a soon-to-graduate student:

> [W]orking on my thesis over the past two months, I have realized that all of the steps I have needed to take have practically been a large comp methods lab. A ton of data cleaning, merging dataframes, converting to dictionaries, looping through requirements of those dictionaries, etc., etc. It has been a hefty task, but there has never been a question of me not knowing what needed to be done thanks to what I had learned during Comp Methods.

But by far, the most powerful and rewarding feedback comes from our graduates who have returned to the Force and are working as analysts in support of real operations. The following message was received in summer 2020 from an officer who had graduated one year earlier and took the time to write about his first assignment after NPS:

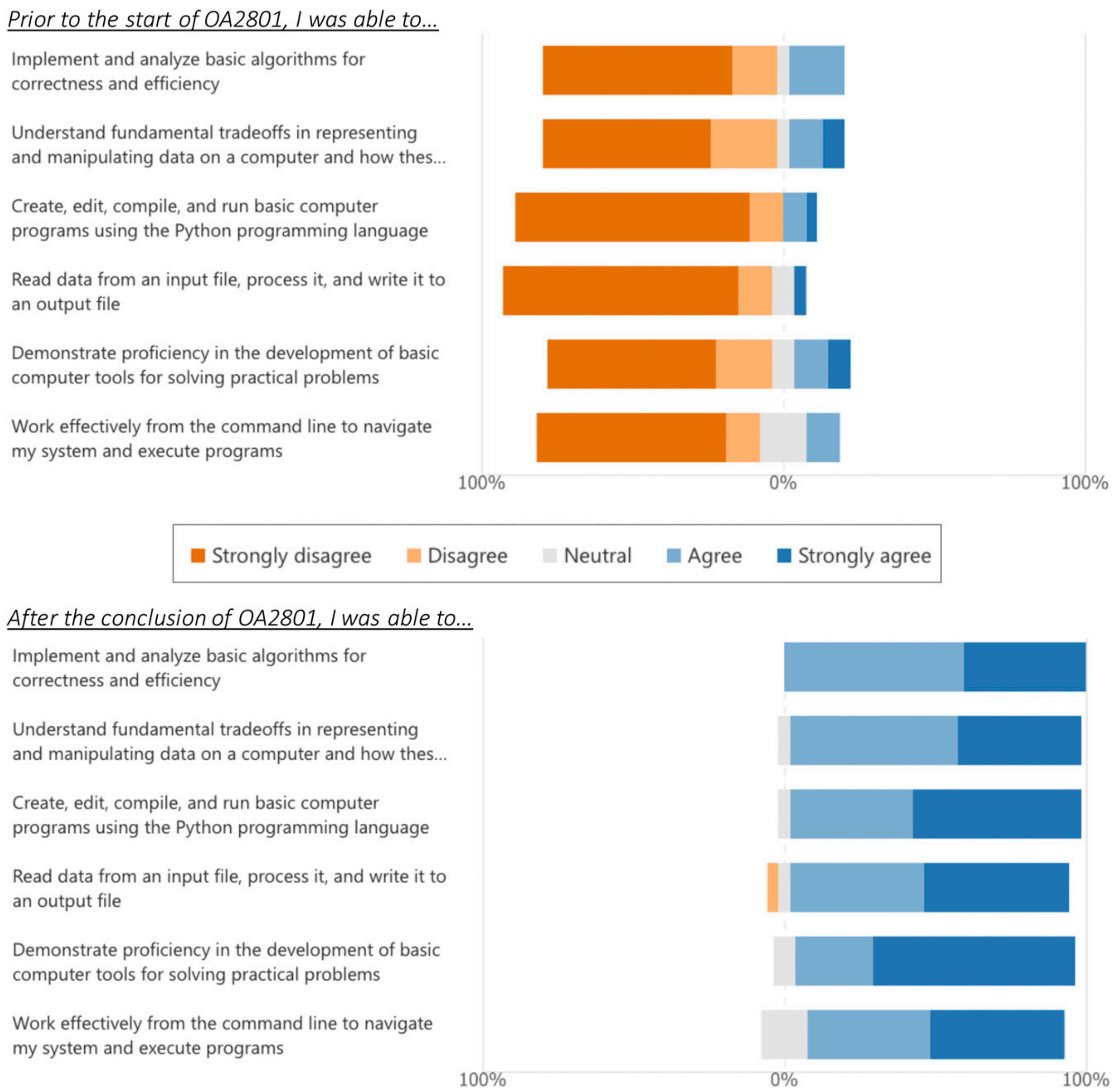
> It's been a huge learning experience for me to say the least. I was originally slated just to serve as an analyst, but the deputy director billet was gapped so I was asked to double fill both roles for my time here. … While I'm still pretty green with the roles, the biggest shock when coming into the workplace was how important speed was to how we do things and deliver products. As an analytics and business intelligence group, we don't typically find ourselves doing a lot of incredibly deep analysis, but it blew my mind (and continues to blow my mind) at how much speed plays into things and how it differentiates us. One of our main functions is to support the operations department and that is a train that never stops moving, sir. It constantly reminds me of the Comp Methods II midterm where we had the live coding portion. I remember we all laughed thinking we would never code against a clock. I kid you not when I say that most days when I log into work, my job feels like that Comp Methods II midterm.

Such correspondence instills additional confidence that we are on the right track, although more work remains.

6. Summary and Conclusion

This paper describes ongoing work at the Naval Postgraduate School to develop and deliver accelerated learning in computation to students who do not have

Figure 5. Self-Assessment of Learning Outcomes in OA2801



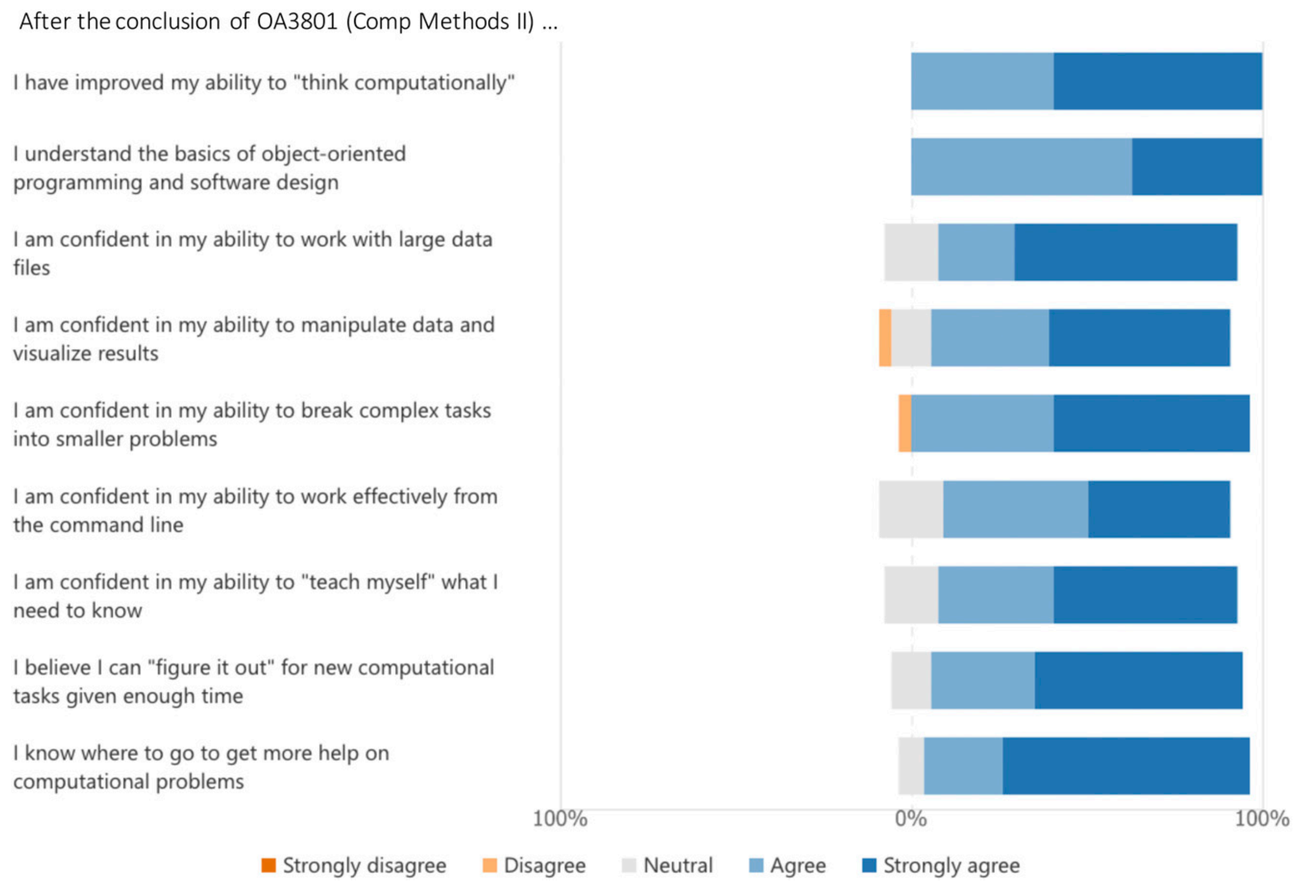
Notes. The questions administered in this survey come directly from the course objectives. Overall, there is a dramatic improvement in student ability after taking this course.

a prior background. A key principle underlying this pedagogy is the need to immerse students in interactive computing exercises both inside and outside of class. This interaction is currently achieved using web browser-based Jupyter notebooks. These notebooks create an encounter that allows the students to actively explore and discover key principles of computation while developing their practical skills. In so doing, students at NPS learn not only to "think computationally" but also "do computationally."

The pace of development within the scientific computing community continues at breakneck speed. Computing libraries and tools that are now considered standard did not exist as little as three to five years ago. This means that our coursework and curriculum must continue to evolve; in practice, we are making significant changes to our courses on a yearly basis. Many of these changes originate with our sponsored research projects that require innovation in the use of these tools—this is graduate education as it ought to be.

Meanwhile, the Jupyter Project itself continues to advance with the introduction of JupyterLab, a new web-based interface that allows the user to combine

Figure 6. Self-Assessment of Computational Ability at the End of the OA2801–OA3801 Sequence



Note. Students report positive outcomes in their ability to think computationally and do computationally.

and arrange individual components (including the Jupyter Notebook, terminal windows, text editors, and more) in a flexible manner (Project Jupyter 2020b). In JupyterLab, a user can customize his or her workflow for developing and executing codes in a manner that is similar to that supported in modern IDEs, such as RStudio (RStudio Team 2015). Understanding how best to use these and other tools for computational doing remains a source of ongoing experimentation.

Acknowledgments

The techniques and best practices described here are the result of considerable experimentation and collaboration on the part of many colleagues at the Naval Postgraduate School, notably Jon Alt, Matt Carlyle, Sam Huddleston, Jeff Hyink, Matt Norton, and Ross Schuchard.

References

Barba LA, Barker LJ, Blank DS, Brown J, Downey AB, George T, Heagy LJ, et al. (2019) *Teaching and Learning with Jupyter*. Accessed February 21, 2020, https://jupyter4edu.github.io/jupyter-edu-book/.

Barr V, Stephenson C (2011) Bringing computational thinking to K-12: What is involved and what is the role of the computer science education community? *ACM Inroads* 2(1):48–54.

Bart AC (2018) Answer to "Create a diverging stacked bar chart in matplotlib." Stack Overflow (forum), December 29, 2016, edited June 21, 2018, https://stackoverflow.com/a/41384812.

Bloom BS, Engelhart MD, Furst EJ, Hill WH, Krathwohl DR (1956) *Taxonomy of Educational Objectives: The Classification of Educational Goals—Handbook I: Cognitive Domain* (David McKay Company, New York).

Brown GG (2004) How to write about operations research. *Phalanx* 37(3):7–13.

Cardoso A, Leitão J, Teixeira C (2018) Using the Jupyter notebook as a tool to support the teaching and learning processes in engineering courses. Auer M, Tsiatsos T, eds. *Proc. 21st Internat. Conf. Interactive Collaborative Learn.* (Springer, Cham, Switzerland), 227–236.

Denning PJ (2002) Career redux. *Comm. ACM* 45(9):21–26.

Denning PJ (2009) The profession of IT beyond computational thinking. *Comm. ACM* 52(6):28–30.

Denning PJ (2017) Remaining trouble spots with computational thinking. *Comm. ACM* 60(6):33–39.

Doumont J (2004) Teaching is not learning: How to make students learn. Presentation at the California Institute of Technology, Pasadena, CA.

Doumont J, ed. (2010) *English Communication for Scientists* (NPG Education, Cambridge, MA).

Doumont J (2014) Teaching is not learning: Going beyond the traditional lecture. Presentation at the Chicago Center for Teaching, September 25, University of Chicago, Chicago. YouTube video, 1:29:05, posted October 9, 2014, by UChicago CCT, https://youtu.be/07QxQRl1viI.

Fenner M (2018) Fernando Pérez and the Project Jupyter Team honored with 2017 ACM Software System Award. *Berkeley Institute for Data Science News* (April 24), https://bids.berkeley.edu/news/perez-project-jupyter-2017-acm-software-system-award.

Glick B, Mache J (2018) Using Jupyter notebooks to learn high-performance computing. *J. Comput. Sci. Colleges* 34(1):180–188.

Grover S, Pea R (2013) Computational thinking in K–12: A review of the state of the field. *Ed. Res.* 42(1):38–43.

Gruber J (2004) Markdown project page. Accessed February 17, 2020, https://daringfireball.net/projects/markdown/.

Guerra H, Gomes LM, Cardoso A (2019) Agile approach to a CS2-based course using the Jupyter notebook in laboratory classes. Cardoso A, Restivo MT, eds. 2019 5th Experiment Internat. Conf. (IEEE, Piscataway, NJ), 177–182.

Guo P (2014) Python is now the most popular introductory teaching language at top U.S. universities. *Comm. ACM* (blog), July 7, https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext.

Hardin JR, Holder A, Beck JC, Furman K, Hanna A, Rader D, Rego C (2012) Recommendations for an undergraduate curriculum at the interface of operations research and computer science. *INFORMS Trans. Ed.* 12(3):117–123.

Hart WE, Watson J-P, Woodruff DL (2011) Pyomo: Modeling and solving mathematical programs in Python. *Math. Programming Comput.* 3(3):219–260.

Hart WE, Laird CD, Watson J-P, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2017) *Pyomo—Optimization Modeling in Python*, 2nd ed. (Springer Science & Business Media, New York).

Hoffman RR (1998) How can expertise be defined? Implications of research from cognitive psychology. Williams R, Faulkner W, Fleck J, eds. *Exploring Expertise* (Palgrave Macmillan, London), 81–100.

Holdgraf C, Culich A, Rokem A, Deniz F, Alegro M, Ushizima D (2017) Portable learning environments for hands-on computational instruction: Using container-and cloud-based technology to teach data science. Proc. Practice Experience Adv. Res. Comput. 2017 Sustainability, Success Impact (ACM, New York), Article 32.

Iverson K (1962) *A Programming Language* (John Wiley & Sons, New York).

Kluyver T, Ragan-Kelley B, Pérez F, Granger BE, Bussonnier M, Frederic J, Kelley K, et al. (2016) Jupyter notebooks—A publishing format for reproducible computational workflows. Loizides F, Schmidt B, eds. *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (IOS Press, Amsterdam), 87–90.

Ko AJ, Abraham R, Beckwith L, Blackwell A, Burnett M, Erwig M, Scaffidi C, et al. (2011) The state of the art in end-user software engineering. *ACM Comput. Surveys* 43(3):1–44.

Koehler JF, Kim S (2018) Interactive classrooms with Jupyter and Python. *Math. Teacher* 111(4):304–308.

Krathwohl DR (2002) A revision of Bloom's taxonomy: An overview. *Theory Into Practice* 41(4):212–218.

Kross S, Guo PJ (2019) Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. *Proc. 2019 CHI Conf. Human Factors Comput. Systems* (ACM, New York).

Likert R (1932) A technique for the measurement of attitudes. *Arch. Psych.* 22(140):1–55.

Lye SY, Koh JHL (2014) Review on teaching and learning of computational thinking through programming: What is next for K-12? *Comput. Human Behav.* 41(December):51–61.

Maestre CR, Gregori FA, López MP, Aldeguer RR (2016) Jupyter Notebook: Theory and practice of mathematical models in engineering and architecture. Gómez Chova L, López Martínez A, Candel Torres I, eds. *Proc. 9th Annual Internat. Conf. Ed. Res. Innovation* (IATED, Valencia, Spain), 6523–6530.

McKinney W (2010) Data structures for statistical computing in Python. van der Walt S, Millman J, eds. *Proc. 9th Python Sci. Conf.* (Austin, TX), 56–61.

McNamara A (2016) On the state of computing in statistics education: Tools for learning and for doing. Preprint, submitted October 1, https://arxiv.org/abs/1610.00984.

Milano F, Jónsdóttir GM (2018) Jupyter notebooks for computer-based laboratories on power system dynamics and control. Gómez Chova L, López Martínez A, Candel Torres I, eds. *10th Internat. Conf. Ed. New Learn. Tech. (EDULEARN 2018) Proc.* (IATED, Valencia, Spain), 112–121.

National Research Council (2010) *Report of a Workshop on the Scope and Nature of Computational Thinking* (National Academies Press, Washington, DC).

National Research Council (2011) *Report of a Workshop on the Pedagogical Aspects of Computational Thinking* (National Academies Press, Washington, DC).

O'Hara K, Blank D, Marshall J (2015) Computational notebooks for AI education. Russell I, Eberle W, eds. *Proc. 28th Internat. Florida Artificial Intelligence Res. Soc. Conf.* (Association for the Advancement of Artificial Intelligence, Menlo Park, CA), 263–268.

Oliphant TE (2006) *A Guide to NumPy*, vol. 1 (Trelgol Publishing USA, Austin, TX).

Papert S (1990) A critique of technocentrism in thinking about the school of the future. Accessed January 23, 2020, http://www.papert.org/articles/ACritiqueofTechnocentrism.html.

Pérez F, Granger BE (2007) IPython: A system for interactive scientific computing. *Comput. Sci. Engrg.* 9(3):21–29.

Pérez F, Granger B (2017) The state of Jupyter: How Project Jupyter got here and where we are headed. *O'Reilly Ideas* (January 26), https://www.oreilly.com/radar/the-state-of-jupyter/.

Perkel JM (2018) Why Jupyter is data scientists' computational notebook of choice. *Nature* 563(7732):145–147.

Perkel JM (2021) Ten computer codes that transformed science. *Nature* 589(7842):344–348.

Poppendieck M, Poppendieck T (2003) *Lean Software Development: An Agile Toolkit* (Addison-Wesley, Boston).

Project Jupyter (2020a) A gallery of interesting Jupyter Notebooks. Accessed January 20, 2020, https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks.

Project Jupyter (2020b) Jupyter Project home page. Accessed January 27, 2020, https://jupyter.org/.

Project Jupyter, Blank DS, Bourgin D, Brown A, Bussonnier M, Frederic J, Granger B, et al. (2019) nbgrader: A tool for creating and grading assignments in the Jupyter notebook. *J. Open Source Ed.* 2(11):32.

Python Software Foundation (2020) Python programming language. Accessed September 20, 2021, www.python.org.

Rao AR, Dave R (2019) Developing hands-on laboratory exercises for teaching stem students the Internet-of-things, cloud computing and blockchain applications. *2019 IEEE Integrated STEM Ed. Conf.* (IEEE, Piscataway, NJ), 191–198.

R Core Team (2019) R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, https://www.R-project.org.

RStudio Team (2015) RStudio: Integrated development environment for R. RStudio, Inc., Boston, http://www.rstudio.com/.

Smith AA (2016) Teaching computer science to biologists and chemists, using Jupyter notebooks: Tutorial presentation. *J. Comput. Sci. Colleges* 32(1):126–128.

Spence R (1975) APL demonstration 1975. YouTube video, 27:59, posted July 12, 2013, by Imperial College London, https://www.youtube.com/watch?v=_DTpQ4Kk2wA.

Suárez A, Alvarez-Feijoo MA, Fernandez Gonzalez R, Arce E (2018) Teaching optimization of manufacturing problems via code components of a Jupyter notebook. *Comput. Appl. Engrg. Ed.* 26 (5):1102–1110.

Swamy V (2018) Pedagogy, infrastructure, and analytics for data science education at scale. Technical Report UCB/EECS-2018-81, Electrical Engineering and Computer Sciences Department, University of California, Berkeley, Berkeley. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-81.html.

Urcelay-Olabarria I, Lazkoz R, Urrestilla J, Leonardo A, Igartua JM (2017) Jupyter Notebook as the physics experimental laboratory's logbook—First approach. Escudeiro P, Costagliola G, Zvacek S, Uhomoibhi J, McLaren BM, eds. *Proc. 9th Internat. Conf. Comput. Supported Ed.*—Vol. 1: CSEDU (SciTePress, Setúbal, Portugal), 458–463. http://dx.doi.org/10.5220/0006352104580463.

Vial G, Negoita B (2018) Teaching programming to non-programmers: The case of Python and Jupyter notebooks. Ram S, Rosemann M, Pries-Heje J, eds. *Proc. 39th Internat. Conf. Inform. Systems*, vol. 4 (Curran Associates, Red Hook, NY), 2832–2848.

Wang Y, Hill KJ, Foley EC (2017) Computer programming with Python for industrial and systems engineers: Perspectives from an instructor and students. *Comput. Appl. Engrg. Ed.* 25 (5):800–811.

Willis A, Charlton P, Hirst T (2020) Developing students' written communication skills with Jupyter notebooks. *Proc. 51st ACM Tech. Sympos. Comput. Sci. Ed.* (ACM, New York), 1089–1095.

Wing JM (2006) Computational thinking. *Comm. ACM* 49(3):33–35.

Wing JM (2008) Computational thinking and thinking about computing. *Philos. Trans. Roy. Soc. A* 366(1881):3717–3725.

Zastre M (2019) Jupyter Notebook in CS1: An experience report. Stephenson B, ed. *Proc. Western Canadian Conf. Comput. Ed.* (ACM, New York), 1–6.

Zhong B, Wang Q, Chen J, Li Y (2016) An exploration of three-dimensional integrated assessment for computational thinking. *J. Ed. Comput. Res.* 53(4):562–590.