Cracking the code: An evidence-based approach to teaching Python in an undergraduate earth science setting

Ethan C. Campbell*[§], Katy M. Christensen*, Mikelle Nuwer, Amrita Ahuja, Owen Boram, Junzhe Liu[†], Reese Miller, Isabelle Osuna, Stephen C. Riser

School of Oceanography, University of Washington, Seattle, Washington 98195

* These authors contributed equally to this work.

Running title: "Teaching Python in an undergraduate earth science setting"
Article type: Curriculum & Instruction
Keywords: Python programming; oceanography; instructional design; active learning; remote teaching

Abstract

Scientific programming has become increasingly essential for manipulating, visualizing, and interpreting the large volumes of data acquired in earth science research. Yet few domain-specific instructional approaches have been documented and assessed for their effectiveness in equipping geoscience undergraduate students with coding and data literacy skills. Here we report on an evidence-based redesign of an introductory Python programming course, taught fully remotely in 2020 in the School of Oceanography at the University of Washington. Key components included a flipped structure, activities infused with active learning, an individualized final research project, and a focus on creating an accessible learning environment. Cloud-based notebooks were used to teach fundamental Python syntax as well as functions from packages widely used in climate-related disciplines. By analyzing quantitative and qualitative student metrics from online learning platforms, surveys, assignments, and a student focus group, we conclude that the instructional design facilitated student learning and supported self-guided scientific inquiry. Students with less or no prior exposure to coding achieved similar success to peers with more previous experience, an outcome likely mediated by high engagement with course resources. We believe that the constructivist approach to teaching introductory programming and data analysis that we present could be broadly applicable across the earth sciences and in other scientific domains.

[§] **Corresponding author:** Ethan C. Campbell, ethancc@uw.edu, School of Oceanography, University of Washington, Seattle, WA 98195, USA

[†] Junzhe Liu is now affiliated with Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY 10964, USA

ORCIDs: *Ethan C. Campbell (https://orcid.org/0000-0002-8588-7506), Katy M. Christensen (https://orcid.org/0000-0003-1064-2245), Mikelle Nuwer (https://orcid.org/0009-0005-2291-8634), Junzhe Liu (https://orcid.org/0000-0002-5538-8992)*

Introduction

Motivation

Data programming has become the foundation of research in today's geoscientific disciplines. As the volume and size of earth science data sets have steadily increased, so have the complexity and ubiquity of the computational techniques used for analysis and visualization. Some argue that innovation in earth science research will increasingly be driven by one's competency in translating ideas into computer code (Jacobs et al., 2016).

The field of oceanography is no exception to this "data tsunami," with more hydrographic casts collected in the past two decades than over the previous 100 years (Brett et al., 2020). Unprecedented collaborative initiatives such as the Argo profiling float array (Wong et al., 2020), the National Science Foundation's Ocean Observatories Initiative (OOI; Greengrove et al., 2020), and remote sensing platforms such as satellite altimeters (Scheick et al., 2023) are continuously adding to expansive, publicly available data sets. In addition to these observational programs, hard drives at institutions across the world are being filled with terabytes of data generated by numerical simulations. From highly resolved ocean general circulation models to lower-resolution global climate models assessed in the Intergovernmental Panel on Climate Change (IPCC) reports, the natural ocean is being reproduced with ever-increasing fidelity (Haine et al., 2021). The resulting challenges in accessing and analyzing these data require new computational tools that enable truly open science, further motivated by the notion that "research conducted openly and transparently leads to better science" (National Academies of Sciences, Engineering, and Medicine, 2018). At the same time, the computational methods used to study the ocean – which have traditionally differed between modeling- and observation-focused oceanographers – remain "radically unstandardized," contributing to scientific code that is influenced by unique requirements and social contexts and may deviate from best practices in software engineering, as highlighted by an ethnography of oceanographers' programming practices (Kuksenok et al., 2017).

Domain-specific computational coursework and data literacy are thus a critical part of a modern oceanographic undergraduate curriculum, and we infer the same applies across many geoscience disciplines. While students can collect and analyze small-scale data sets through hands-on fieldwork and labs that are common elements of undergraduate earth science curricula, working with larger, professionally collected data sets requires familiarity with a programming language (Kastens et al., 2015). Historically, introductory programming education has been the responsibility of computer science departments, with a focus on data structures and algorithms. Geoscience-specific programming instruction will necessarily reflect distinct goals and tools compared to computer science (Grapenthin, 2011) or data science (Anderson et al., 2015; Lasser et al., 2021), namely, the use of coding to derive insight into natural systems through mathematical manipulation, visualization, and interpretation of idiosyncratic data, often in the time and space domains. Yet scientific computing is often absent in earth science curricula, including oceanography (Old, 2019), except for highly scaffolded coding modules in courses where programming is not the focus (e.g., Rowe et al., 2021). In this void, brief but intensive hands-on workshops like those offered by Software Carpentry (https://software-carpentry.org; Wilson, 2016), Data Carpentry (https://datacarpentry.org/; Irving, 2019), and scientific societies (e.g., Arms et al., 2020) have provided crucial training to young scientists. These short workshops, however, give learners limited opportunities to apply new coding skills to their own research in a supervised setting. In lieu of formalized instruction, many earth science students teach themselves programming during research experiences or in graduate programs, which can lead to the propagation of ad hoc, inefficient, and outdated practices.

Incorporating programming into an earth science curriculum additionally opens the door to a constructivist approach to teaching scientific concepts—one that encourages students to use experimentation and individualized, self-guided inquiry to build on previous learning, construct new knowledge, and engage in critical reflection (Bada, 2015; Hadjerrouit, 2008). The iterative, reflective process of writing and refining scientific code makes it naturally suited to this individualized model of learning. In practice, a constructivist pedagogy – much like programming instruction – often involves active techniques such as project-based investigation, cooperative learning, and inquiry-based activities, which have been shown to improve student competencies in information recall, analysis, and quantitative reasoning in the context of a large-enrollment introductory oceanography course (Yuretich et al., 2001).

Throughout higher education, there is an increasing recognition that effective teaching requires a focus on active learning, which can be described broadly as "any instructional method that engages students in the learning process" (Prince, 2004). Active modalities stand in contrast to traditional lecturing, which represents about three-quarters of class time across STEM undergraduate and graduate courses today (Stains et al., 2018). There is strong evidence that using active learning techniques increases student performance – that is, students' understanding and retention of material – in STEM courses, with disproportionate benefits for underrepresented students and students who learn in different ways (Freeman et al., 2014; Haak et al., 2011; Theobald et al., 2020). One reason these strategies appear to be effective is that they often require an instructor to implement more structure in their course through, for example, regular and intensive practice using scaffolded activities (Haak et al., 2011). Evidence supporting the efficacy of active learning strategies in geoscience classrooms is more limited due to a paucity of discipline-specific research, but a variety of easily implemented student-centered activities and techniques have been documented (McConnell et al., 2017).

Embedding computing skills into a geoscience curriculum faces the challenge of introducing students to unfamiliar skills such as algorithmic thinking and overcoming a steep learning curve, similar to teaching a foreign language (Jacobs et al., 2016). Perhaps for this reason – as well as a lack of accessible software tools and computational power in previous decades (Hays et al., 2000) – existing examples of courses using geoscience data have often focused on interactive online modules, portals, or widgets that are constrained in their data sets and capabilities (e.g., Ellwein et al., 2014; Greengrove et al., 2020; Klug et al., 2017). Software such as Microsoft Excel or specialized tools like Ocean Data View face similar limitations. In comparison, programming skills are more versatile, enabling the analysis of virtually any data set from any domain and empowering students to conduct independent or mentored research projects.

Why teach Python?

In an introductory classroom setting, the choice of programming language matters. Python is an ideal candidate, as it is easy to learn, versatile, and free to use. First released three decades ago, Python is increasingly ubiquitous within earth science (Lin, 2012) and is widely used outside the scientific community, particularly in industry, making it valuable even for students seeking a career outside of academia (Srinath, 2017). The language features concise, easily read, higher-level syntax that allows one to focus on data exploration, enabling more efficient science (Ayer et al., 2014; Jacobs et al., 2016; Lin, 2012). For those learning programming for the first time, a primary challenge is thinking algorithmically, that is, developing structured code to solve a problem. Compared to Python, lower-level programming languages commonly taught in introductory computer science courses (such as Java and C++) require substantial syntactical overhead that can distract from achieving that pedagogical goal (Pears et al., 2007; Srinath, 2017).

Python offers other advantages (Gentemann et al., 2021). Its open-source nature has fostered a large active developer community, which has contributed to its stability and the dissemination of numerous multipurpose packages that extend its functionality. Python is free to download and use, avoiding reliance on expensive commercial solutions that can render analysis code inaccessible to scientists outside of well-resourced university environments. These stand in contrast to MATLAB, a scientific programming language also popular in geoscientific research. Despite the clear benefits of teaching Python in an earth science context, we find only one documented example of an instructional approach for a quarter- or semester-long course in the existing literature (Jacobs et al., 2016).

Course history and development

Our study reports on an evidence-based redesign of an undergraduate oceanography course that teaches introductory Python data analysis techniques. In subsequent sections, we highlight key course elements (summarized schematically in **Fig. 1**) and assess the efficacy of the redesign from the standpoint of student engagement and learning.

S.C.R. established and previously taught "Methods of oceanographic data analysis" (OCEAN 215) annually in the School of Oceanography at the University of Washington from 2015-2019. It was the first introductory Python course offered by the department and met in person two times each week in two-hour sessions that featured a mix of traditional lecturing and dedicated homework time. Over a ten-week quarter, students completed four assignments using programming techniques taught in lectures. The course was well-received by students, who rated it as "very good" (4 on a scale from 1-5) across a variety of metrics in end-of-quarter evaluations from 2015, 2016, 2017, and 2019 (**Fig. 2**), and has been perceived as demanding relative to other courses in students' curricula (see **Fig. S1** in Supplemental Materials).

However, faculty teaching other courses in the department's curriculum reported that many students who completed OCEAN 215 had difficulty with core Python programming tasks. A review of past senior theses – projects in which students formulate and execute original research – revealed that students often used minimal scientific code and reverted to less versatile, non-coding solutions like Microsoft Excel and Google Earth for data visualizations, to the detriment of their science. Given that students recognized the usefulness of the course content after completing the course (see **Fig. S1** in Supplemental Materials), we partially attribute their subsequent hesitancy and lack of confidence in applying Python skills to weaknesses in the course design, some of which are prevalent across undergraduate education:

- *An overreliance on non-interactive lectures.* This is commonplace—in a survey of almost 200 undergraduate oceanography professors, for example, three-quarters indicated that they use data in their teaching but are most likely to use a lecture teaching strategy, rather than creating opportunities for active inquiry (McDonnell et al., 2015). As detailed above (see Introduction section "Active learning"), traditional lecturing is less effective at promoting student understanding and retention of material than active learning techniques.

- *A lack of student-driven inquiry.* In assignments, students answered prescribed questions and worked with tidy, unrealistically clean scientific data. Such a controlled environment is valuable for practicing basic skills but offers students few opportunities to pose their own questions and engage in "open inquiry," which Banchi & Bell (2008) associate with deeper, more original scientific thinking.

- *A stagnation of curriculum.* Since the course's launch in 2015, the scientific computing landscape has rapidly evolved (Gentemann et al., 2021). However, certain course elements not reflective of current scientific Python practices were still taught, resulting in the use of outdated, unsupported, and unnecessarily limiting packages and methods. At the same time, the course did not formally address essential programming practices such as commenting etiquette, formulaic code debugging, and use of online documentation.

The course was restructured (**Fig. 1**, **Table 1**) and subsequently co-taught during a 10-week quarter in 2020 by two graduate students (E.C.C. and K.M.C.), both of whom had served as TAs in past years. Twenty-five undergraduate students completed the course, a typical class size (**Fig. 2**). The plurality were third-year oceanography majors. No prior knowledge of computing or upper-level math was required or assumed. Elements retained from previous iterations included the basic format of four structured programming assignments as well as twice-weekly classes and office hours; however, the latter were conducted virtually rather than in a physical classroom space.

In 2020, the COVID-19 pandemic forced a swift transition to virtual instruction. The timing of this course in Autumn 2020, however, allowed for careful planning of an online learning framework, rather than the forced adoption of emergency remote instruction necessary in the first half of 2020 (Donham et al., 2022; Hodges et al., 2020). Nonetheless, disruptions outside of the classroom were still present: students dealt with being isolated on campus or sequestered at home with family, research programs had to be reconfigured, mental health declined, and many became sick or had loved ones fall ill or even pass away (Furman & Moldwin, 2021). With these realities in mind, the course redesign also paid special attention to the need for a supportive and accommodating learning environment (Shay & Pohan, 2021).

The updates to the course were guided by past experience as TAs, consultation with previous teaching teams and department faculty, the need for fully virtual instruction during the COVID-19 pandemic, and a desire to infuse the course with active learning strategies. Changes included flipped video lessons delivered on the online platform Panopto, an individually-driven final research project, content that reflected the current scientific Python ecosystem (including cloud-based notebooks; see **Table 1**), discussions on the online question-and-answer (Q&A) forum Piazza, analysis of data from a wider range of earth science domains, encouragement of pair collaboration and use of external resources, and a syllabus with explicit policies, expectations, and the following end-of-quarter student learning outcomes:

- Understand why the Python programming language is ideal for data analysis.
- Write, execute, and debug Python code.
- Access, read, transform, visualize, and interpret oceanographic data with confidence using Python.
- Explore the ever-expanding universe of packages and tools available for creating and sharing code.
- Formulate and investigate scientific research questions using programming and data analysis skills.
- Adopt best practices in programming and data visualization that facilitate collaboration and information-sharing, both within the classroom and the broader scientific community.

All course materials were original, created by the graduate instructors, and are available for free reuse and adaptation under a CC-BY-4.0 license at https://ethan-campbell.github.io/OCEAN_215/.

Methods

We qualitatively assess the effectiveness of instructional approaches in Autumn 2020 using descriptive examples from the quarter. We also quantitatively analyze the data from standardized course evaluations, an end-of-quarter student survey, graded assessments, and engagement/usage metrics provided by the video and Q&A platforms. Various student-specific engagement and performance metrics were collected by the co-instructors (E.C.C. and K.M.C.), as described in sections below. Prior to analysis, all metrics were de-identified and coded by a coauthor (M.N.) who was not directly involved in quantitative analyses; identified versions were not used thereafter. This study was approved as qualifying for exempt status for institutional review by the Human Subjects Division at the University of Washington.

Initial, mid-quarter, and end-of-quarter surveys

To gauge initial exposure to the Python programming language and to coding in general, students were asked to share their prior experience(s) in an introductory survey issued during week 1 (Assignment #0). The instructors translated students' short-answer responses into a numeric rating (1-5) using a subjective analysis of their word choice (see rubric in **Table S1** in Supplemental Materials). The factors considered were any previous coding languages learned, the reported efficacy of past learning experiences, and time since last exposure to coding.

We also obtained summary reports from end-of-quarter Instructional Assessment System (IAS) surveys completed by OCEAN 215 students in 2015, 2016, 2017, 2019, and 2020 (results from 2018 were unavailable), which were administered and anonymized by the University of Washington. Standardized questions asked students to evaluate aspects of the course quality and their engagement with the course. While most questions were consistent across years, others evolved in their wording and thus required mapping or aggregation to enable comparison between years (as shown in **Table S2** in Supplemental Materials). Questions that could not be tracked across years were excluded. Students completed surveys either in paper or online format, with the class response rate of around 70% in 2020 being somewhat higher than in past years (**Fig. S1** in Supplemental Materials). As IAS summary reports correspond to specific instructors, we averaged the class median responses between the two graduate instructors for each question in 2020.

Furthermore, we referenced students' anonymous responses to open-ended questions from two IAS surveys in 2020: a mid-quarter evaluation administered during weeks 4-5 of the course and the final evaluation. The survey prompts are listed in **Table S3** in the Supplemental Materials. In addition to excerpting quotes from students' responses, we identified common or unique themes mentioned by students and tabulated the frequency with which each theme was mentioned in either a subjectively positive context (e.g., an appreciative or affirming comment; assigned a value of +1) or subjectively negative context (e.g., an unenthusiastic or critical comment; assigned a value of –1) (**Fig. 3)**.

In addition to the university-managed IAS surveys, a Google Form survey was administered during the week after the final class to measure students' perceived success relative to the main objectives outlined in the syllabus. The response rate was 92%. Submissions were not anonymous, but instructors guaranteed that students' responses would not impact their final course grades. As a final self-assessment of students' Python skills, we use responses to the question, "How proficient do you feel in writing, executing, and debugging Python code?", which were on a 6-point scale from "Least proficient" to "Most proficient."

Flipped video viewership

Panopto, the course video hosting and delivery platform, provides instructors with usage statistics, including view counts, minutes delivered, percent completed, and last view time. Those metrics – associated with individual students, individual videos (both aggregated and disaggregated by student), and distinct video viewing sessions, where applicable – were downloaded, and student identities were anonymized as described above. Usage data are presented in **Fig. 4, Fig. 5a**, and **Fig. S2** in the Supplemental Materials. Student-specific Panopto metrics computed for **Fig. 6** include total minutes watched, minutes watched before the class for which a video was assigned, and minutes watched after class for the first time (i.e., late views).

Final grades and programming skills

To measure learning outcomes, students' final grades and programming skills at the conclusion of the course are presented in **Fig. 6**. Grades were recalculated to ignore assignments that students did not complete (i.e., dropping grades of 0%), and the following weights were re-applied: 60% for assignments #0-#4 (weighted equally), 15% for Piazza posts, and 25% for final projects. Original and recalculated final grades averaged 95.0% and 95.9%, respectively, with standard deviations of 5.7% and 3.8%. Programming skills were evaluated as the fraction of Python syntax (functions, operators, and methods) taught in the course that were used at least once in each student's final project code notebook (see **Table S4** in the Supplemental Materials). This metric varies widely between students from 6% to 29% of all syntax keywords taught and thus offers significant discriminatory power, albeit limited by our exclusion of miscellaneous functions that were not taught in the course but were used by some students at higher skill levels.

Online forum engagement

Piazza, the online Q&A platform, also makes usage statistics available to instructors. The following student-specific metrics (presented in **Fig. 6**) were downloaded, then anonymized as described above: days online, answers, and total contributions (which include questions, notes, answers, and comments). Additionally, a time series of engagement was constructed (**Fig. 5a**) based on unique users per day, as provided by Piazza. The time series was supplemented by a manual tabulation of daily Piazza activity within the following categories: student questions and notes related to programming; student scheduling, extension, or logistical requests; student answers and comments; student posts that were required for assignments; and instructor posts, answers, or comments. Where relevant, those categories were further divided by chosen audience into total posts that were public and signed, public and anonymous, or private (i.e., visible to instructors only), as shown in **Fig. 5b**.

Student focus group

Undergraduate students who completed OCEAN 215 in Autumn 2020 were considered for a focus group based on responses to a voluntary survey asking students to rate their interest in the project and to provide a short paragraph about course elements that affected their learning positively or negatively. Five students were chosen by E.C.C. and K.M.C. based on the thoughtfulness of their written responses and the diversity of their academic backgrounds and experiences within the course. Selection was not dependent on students' grades in the course, and it was made clear that survey responses would not impact course grades in any way (and in fact final grades were issued over a month prior to selecting students). Three focus group sessions were held in the quarter following Autumn 2020, each lasting 1-2 hours. In the sessions, E.C.C. and K.M.C. asked questions designed to provoke open and candid discussion on students' perception of course elements. Insights gleaned from the focus group are clearly denoted in the text. We use them as supporting evidence to depict students' perspectives about the course more holistically and accurately, and to indicate areas where students felt the course could be modified to improve their experience.

Additionally, at the request of E.C.C. and K.M.C., four of the five students shared short testimonials detailing their unique experiences in the course, which are presented in **Box 1**. The testimonials were assembled from students' responses to their selection of a subset of the guiding questions included as **Table S5** in the Supplemental Materials and were edited for style and grammar. As noted below in Author Contributions, the five undergraduate students were offered coauthorship on the basis of their substantive intellectual and written contributions to this study and were full participants in providing input on the final manuscript. The undergraduate student coauthors did not have access to the anonymized student metrics described above and did not participate in analysis of the data.

Course elements

Course content

OCEAN 215 taught scientific Python skills needed for oceanographic data analysis, starting with fundamental Python syntax, as well as data management and research practices (**Table 1**). Students learned core functions (see **Table S4** in Supplemental Materials) from versatile, interoperable, and open-source software libraries widely used in climate-related disciplines: NumPy, a fundamental library for multidimensional array computing (Harris et al., 2020); Matplotlib, a visualization library (Hunter, 2007); Cartopy, a mapping toolbox (Met Office, 2022); SciPy, a scientific and statistical analysis library (Virtanen et al., 2020); Pandas, a toolkit for working with 1-D and 2-D data (McKinney, 2010); and Xarray, a toolkit for label-based, coordinate-aligned manipulation of multidimensional netCDF data files (Hoyer & Hamman, 2017). Students were encouraged to reference online documentation and use their knowledge of general function syntax to expand their Python capabilities beyond the course content. Lessons also addressed programming best practices, such as modularizing code, adhering to variable naming conventions, writing comments, and applying consistent style and formatting (Wilson et al., 2014), as well as effective visualization principles, including legibility and labeling (Hepworth et al., 2020) and considerations of accuracy and accessibility when choosing colormaps for visualizations (Thyng et al., 2016). These concepts were introduced using examples and data from oceanographic disciplines (physics, chemistry, biology, and marine geology) and other domains (e.g., cryosphere, atmosphere, and climate) using scaffolding to familiarize students with new topics.

That said, the most novel aspect of this course was not its content but rather how it was taught. As we discuss in the following sections, an effective learning environment was created through the use of evidence-based pedagogical elements: a mix of flipped lectures and engaging activities, opportunities for student collaboration, an online discussion forum, a student-designed research project, and efforts to center accessibility and foster classroom community.

Google Colab notebooks

Google Colaboratory (Colab), a cloud-based, in-browser Python development environment modeled after Jupyter notebooks, was chosen as the coding platform for the course. Notebooks can include a mix of interactive code blocks and narrative text, allowing for easy exploration of data and documentation of scientific workflows. Jupyter notebooks are widely used and considered one of the top 10 computing advances that have transformed science (Granger & Pérez, 2021; Perkel, 2021). In general, cloud-based computing has democratized the ability to conduct complex analyses of earth science data sets, and have created new opportunities for innovation, transparency, and reproducibility (Gentemann et al., 2021).

Google Colab is an ideal teaching platform compared to alternatives like an integrated development environment (IDE) and Jupyter notebooks. Unlike IDEs, Colab requires no local installation of Python or additional software, so students could start coding immediately with minimal device-specific troubleshooting. Notebooks also avoid the cognitive overhead associated with learning command-line syntax or a professional-level IDE (Jacobs et al., 2016; Pears et al., 2007). Unlike Jupyter notebooks, Colab does not require server configuration and integrates with Google Drive, facilitating file sharing and submission of assignments. Comments can be added to notebooks for grading purposes, similar to Google Docs, and built-in edit history can confirm students' compliance with deadlines. While constraints exist, such as a lack of transparent package management, computational limitations, and the need for an internet connection, the advantages of Google Colab outweigh its disadvantages in a classroom setting.

Flipped structure

Blended learning models have been shown in a systematic review to improve the learning experience of novice programmers, as they allow class time to be reserved for active learning and afford students more flexibility to plan and customize their study (Alammary, 2019). In our course, a flipped classroom approach was implemented by assigning 14 recorded lessons of approximately 30 minutes each to be watched before synchronous (Zoom) sessions. Most lessons consisted of lectures using slides that illustrated Python concepts using multiple representations, which has been suggested as a core pedagogical strategy for teaching programming (Hadjerrouit, 2008). For example, slides introducing a new concept would often include three distinct representations: a simplified overview of syntax and function arguments, a minimal example of the function or concept being used (e.g., **Fig. 1b**), and a schematic or illustrative plot. Consistent fonts, color schemes, and other design elements were used to reliably indicate relationships between concepts and distinguish examples from core syntax. Some lessons used live-coding demonstrations rather than slides. Accompanying Colab notebooks were provided with each lesson to allow students to run code while watching.

The 14 flipped lessons were divided into 41 tightly scripted segments of about 10 minutes each (see **Fig. S2c** in Supplemental Materials). This was done with the goal of helping students maintain focus, as some evidence suggests the average student has an attention span of 15–20 min during traditional lecturing (Middendorf & Kalish, 1996). In addition to segmenting videos, students were reminded to take breaks between segments. Students in the focus group indicated that they indeed used these opportunities to step away and refocus. While one student reported in their final course evaluation that "occasionally the length of the recorded lectures prevented [them] from finishing them entirely," we find no significant correlation between video or lesson duration and fraction watched (see **Fig. S2f, Fig. S2h** in Supplemental Materials).

In total, students spent 166 hours watching lesson videos on the Panopto platform. Two-thirds of the watch time occurred before the class for which the video was assigned (**Fig. 4**). Most lessons were released 1.5-3 days before the Zoom class meeting, and students generally watched lessons during the 24 hours prior to class. The remaining one-third of total watch time occurred throughout the month following the relevant class, of which three-quarters were first-time views. This indicates that some students attended class without having watched videos, but did so later, perhaps while completing assignments. Students in the focus group expressed that they appreciated the opportunity to watch videos at a convenient time. Some shared that they would have viewed videos immediately before class regardless of release timing, while others said they would have taken advantage of a longer period of availability. Half of students watched nearly every video, with class-wide average video completion between 80-90% in most weeks (**Fig. 5a**). Completion rates dropped near the end of the course, which student focus group participants suggested was due to high end-of-quarter demands in other courses and because the material covered didn't appear in assignments.

The flipped structure appears to have enabled a diversity of strategies for content acquisition. Some students in the focus group re-watched videos to review material or used corresponding slide decks for the same purpose, while another student took notes on the videos and later referenced those notes. In final course evaluations, students noted that having slide decks available benefitted their learning (**Fig. 3**), with one student sharing, "I was able to surprise myself with how much I could figure out through review when feeling helpless at first." Despite the addition of watching flipped videos (as well as a final project) to the overall course workload, students reported in final evaluations that the amount of time they spent each week was similar to past quarters. Yet students reported that out of the total time spent on the course, a greater fraction than in past quarters – nearly 90% – was valuable in advancing their education, and that their participation was higher (**Fig. 2**). In line with prior research on the student perspective of the flipped model (McCallum et al., 2015), our course structure generally received students' approval in course evaluations (**Fig. 3**).

Synchronous class sessions

In-class sessions were conducted using the Zoom platform. Each synchronous class started with simple icebreakers and anonymous Poll Everywhere polls to gather feedback about previous video lessons. Following these activities, concepts from the relevant flipped videos were briefly reviewed, with ample time for students to ask lingering questions. In some class sessions, short activities were used to introduce topics not covered in lesson videos. Classes often concluded with discussions of course logistics and upcoming deadlines. One-on-one tutoring was offered in lieu of class sessions for students located in remote time zones, among other accommodations (see Course Elements section “Accessibility and inclusivity”).

The majority of synchronous class time on Zoom was spent facilitating coding tutorials that integrated concepts taught in the video lessons. Tutorials were designed with multiple goals in mind, in alignment with core considerations for programming activities laid out by Hadjerrouit (2008): (1) to encourage students to analyze the problem at hand and develop stepwise solutions to address separate components; (2) to build on concepts that students previously learned, encouraging reuse and modification of previous code examples; and (3) to compare and contrast different ways of achieving the same analytical or graphical result. The purpose of class activities was clearly communicated to students to explain why they were relevant.

Tutorials were presented in a Google Colab notebook for each class, which students would copy within the Google Drive file structure so that they could edit their notebook individually. In each notebook, copious scaffolding around each problem (e.g., step-by-step instructions, expected intermediate results, and links to documentation websites) was often provided to create an environment of “structured inquiry.” In the hierarchy of Banchi & Bell (2008), who propose a four-level continuum of inquiry, for example, structured inquiry represents the second level, followed by the more independent modes of “guided inquiry” and “open inquiry.”

A tutorial notebook would often include four or five related but distinct problems that applied different concepts or functions to a real-world data set from oceanographic and related disciplines (e.g., **Fig. 1c**); data were curated by the instructors for their instructional potential. These exercises created opportunities to divide the classroom into small groups that worked cooperatively within Zoom breakout rooms. A modified "think-pair-share" model (McConnell et al., 2017; Yuretich et al., 2001) was adopted: students first individually attempted a problem for a few minutes, then teamed up with their group of classmates in a breakout room to discuss challenges encountered and optimal solutions, and lastly returned to the main Zoom room, at which point a designated 'reporter' from each group reviewed their results with the full class. Instructors monitored student discussions by moving between breakout rooms and providing guidance when needed. Groups' progress was tracked by watching a shared Google Doc configured ahead of time with templates in which each group was told to fill in their code after they finished their work. We recommend that instructors consider randomizing groups occasionally so that students get exposure to a variety of coding styles, social dynamics, and levels of confidence with the material.

Student focus group participants shared mixed views on the number of students per group, as smaller groups require more individual accountability, but larger groups allow instructors cycling between breakout rooms to provide more efficient guidance. Additional benefits of larger groups include increased opportunities for peer instruction and a higher likelihood of at least one student having the required understanding to assist their group in completing an activity. In course evaluations, students mostly offered criticism on the use of breakout groups, with one noting, "I didn't find the small group coding breakout rooms very helpful for coding, but they were nice for getting to know my classmates." While breakout rooms allow for more individualized attention, instructors must be careful to distribute their finite time across groups. Several students wished for more time and instructor guidance in breakout rooms, which contributed to their overall negative rating (**Fig. 3**).

On the other hand, interactive tutorials involving live coding demonstrations and individual activities were the most positively reviewed course element in students' mid-quarter and final surveys (**Fig. 3**). Based on the mid-quarter feedback, the instructors emphasized these tutorials and live coding in the second half of the course. Compared to using slides or copying and pasting blocks of existing code, live coding offers several advantages: it forces slower, more digestible instruction, allows instructors to be responsive to student questions in real-time, and inevitably allows students to see instructors' mistakes and how they are diagnosed and fixed (Wilson, 2016).

The unique challenges posed by virtual teaching require instructors to explore alternative avenues of assessing student understanding. Opportunities for engagement were provided through breakout rooms and use of the chat function to ask and answer questions; in final course evaluations, students rated their participation as higher relative to other courses (6.0 on a 7-point scale, where 4.0 is "average"; **Fig. 2**).

Assignments

Students completed four programming assignments at two-week intervals, each consisting of approachable, multi-part problems in a Google Colab notebook that utilized real scientific data (e.g., **Fig. 1d**). For example, one assignment tasked students with importing data collected by an ocean observing platform (a seaglider), identifying key summary statistics, creating a visualization of the glider's location and temperature measurements, and calculating trends in the data.

Assignments incorporated elements of both "structured inquiry" and "guided inquiry," the second and third levels in the hierarchy of Banchi & Bell (2008). Questions were somewhat less structured than in class activities, allowing students more flexibility to design their own solutions. This created opportunities to practice both programming skills and data literacy, creating a stepping stone to more sophisticated independent analysis of data sets. Without a midterm exam, assignments were instructors' main window into student progress prior to the final project. The assignments were designed to be challenging yet were viewed favorably by both the student focus group and the final evaluation respondents (**Fig. 3**). Both, however, indicated a desire for more short, frequent, low-stakes practice opportunities to help reinforce concepts and check understanding.

Pair programming

Students were offered the option to collaborate in pairs on assignments and the final project, which 48% of the class exercised at some point and, on average, 37% of students exercised on any given assignment. The number of times that a student worked collaboratively is presented as the metric "Pair programming experiences" in **Fig. 6**. When programming as a pair, one student may serve as the "driver," writing code, while the other observes, monitoring the code for defects and helping to problem-solve. Pair programming has long been known to improve student learning, performance, and satisfaction in the computer science classroom, without loss of competency on exams (e.g., McDowell et al., 2002; Williams & Upchurch, 2001). Previous work has found equal benefits to student performance and confidence for students who pair program remotely using screen-sharing and audio connectivity compared to physically collocated students who pair program (Hanks, 2005). In a survey of undergraduates who conducted collaborative research, almost 80% reported that working in teams or pairs enhanced their research experience (Lopatto, 2010).

We found pair programming to be readily adaptable to the virtual classroom using Zoom screen-sharing, with the caveat that Colab notebooks must be refreshed to show updates and thus edits must be made by one user at a time rather than synchronously. One lesson learned was that some pairs will gravitate towards asynchronous collaboration (i.e., a division of labor, rather than true pair programming) unless it is specified that the coding must be done synchronously. Additionally, collaborations appeared to prove more successful when coding partners had a pre-existing working relationship; naturally, this is less likely to occur in a remotely taught introductory class setting.

Piazza

In the context of a pandemic that saw many undergraduate students isolated from friends and support networks, there was an urgent need to cultivate a classroom community. An online Q&A board, Piazza, was offered as an outlet for students to connect asynchronously with peers and instructors outside of class and office hours (see **Fig. 1e**; we note that alternative platforms with similar functionality exist, e.g., Ed Discussions). Instructors benefit from receiving fewer individual emails from students and being able to endorse student answers. Students benefit from easier access to help – not only on logistical or clarifying questions, but also when seeking support on their problem-solving processes. Previous study in an undergraduate computer science setting found that students use Piazza for this full range of question types (Vellukunnel et al., 2017). This past work notes that asking a question on a discussion forum, by definition, constitutes a form of active learning, though posts may vary in their level of reasoning and connectedness.

We find that engagement with Piazza in the form of questions, answers, and comments closely tracked assignment deadlines and peaked while students worked on the final project (**Fig. 5a**). Many questions from students were simple – for example, diagnosing a coding bug or clarifying the goal of an assignment – while others were more complex – such as seeking strategies to efficiently work with large data sets for one's final project. Four brief check-ins (including Assignment #0) required Piazza submissions and an additional quota of five substantive posts per student (i.e., those that contribute "further insight" to the discussion, rather than simply writing "Good work" or "I agree") was prescribed in the syllabus. That said, voluntary engagement was unexpectedly robust, with students visiting Piazza once every 1-5 days on average. The forum saw 889 total contributions, out of which two-thirds of students' posts were not required by a check-in or Assignment #0 (**Fig. 5b**). Past work has likewise shown high participation rates on Piazza when students are encouraged to use the platform by teaching staff (Vellukunnel et al., 2017).

In the ideal case, Piazza would be used by students to seek help after they have invested time into trying different solutions and have perhaps consulted online resources, rather than as an option of first resort. The asynchronous nature of the forum also encourages students to look elsewhere first. While prompt instructor engagement is vital for establishing a strong teaching presence in a remotely taught course (Prince et al., 2020), it is important that responses be somewhat delayed so that an expectation of near-instantaneous feedback is not established. Importantly, this also allows peers an opportunity to provide input. Nonetheless, the instructors found that delaying feedback – particularly when a question had a straightforward answer – often ran against their desire to help students, and thus proved challenging.

The platform allowed students to select the audience for their questions (instructors and/or classmates), to post anonymously, and to respond to peers in threaded discussions. Students selected the three audience options (public, signed or anonymous, and private posts) with approximately equal frequency, depending on their needs

(**Fig. 5b**). Student focus group participants shared that the anonymous and private posting options were useful when they were worried that a question would be perceived as obvious or simple, or when they were less sure of their answer. Final course evaluations show that students felt positively about having access to Piazza (**Fig. 3**). One student shared their appreciation for the ability to post anonymously, stating that it "alleviated some anxiety about asking questions."

Final project

Students completed an individually-driven or collaborative final project. The goal was for students to write code to explore a scientific data set of their choice, supported by ample guidance from the instructors, peer review from classmates, and use of external resources. Similar to the structure of an introductory data programming course described by Anderson et al. (2015), low-stakes checkpoints throughout the quarter required students to share their topic, data set, scientific questions, and hypotheses on the Piazza Q&A board, as well as offer feedback on at least three other classmates' choice of data or questions. The project culminated in each student or pair delivering a short final presentation. A rubric was provided to clearly communicate expectations and evaluation techniques for code, figures, and presentation content and delivery (see **Table S6** in Supplemental Materials). A literature review tentatively indicates that rubrics can lead to increased student performance, and in any case, rubrics are recognized as a user-friendly tool for setting guidelines and enabling self-assessment (Brookhart & Chen, 2015).

In contrast to instructor-generated activities, the final project allowed for student-designed questions and procedures. This encouraged "open inquiry" – the highest level of the hierarchy presented by Banchi & Bell (2008) – an experience that is exceedingly rare in undergraduate oceanography teaching (McDonnell et al., 2015). In general, inquiry-based learning develops cognitive skills on higher levels of Bloom's taxonomy (Bloom et al., 1956; Krathwohl, 2002). Consistent with a constructivist approach to learning (Bada, 2015), the project exposed students to complex or potentially ill-structured questions and 'messy' real-world data sets that were flawed or incomplete (e.g., Ellwein et al., 2014; Klug et al., 2017), though instructors offered guidance related to feasibility. In courses where undergraduate students conduct research with unknown outcomes, students have reported learning gains similar to those of dedicated summer research programs (Lopatto, 2010). In final course evaluations, most students viewed the final project as beneficial, specifically citing the opportunity to synthesize course knowledge and to collaborate with classmates (**Fig. 3**). One critical comment related to ambiguity about the rigor of science expected and the open-ended nature of project checkpoints.

The final projects that students produced were impressive and original, and spanned oceanographic, cryosphere, and atmospheric domains (see **Fig. S3** in Supplemental Materials). Here we assess students' final project questions and hypotheses based on four higher levels of the cognitive process dimension of the revised Bloom's taxonomy (Bloom et al., 1956; Krathwohl, 2002), namely application, analysis, evaluation, and creation (see rubric in **Table 2**), similar to the methodology of Kastens et al. (2020). We also evaluate each project's complexity by summing the number of scientific domains, file types, and data sets incorporated. We find that students' project cognitive levels were consistent between the questions and hypotheses they posed. Interestingly, we identify no significant relationship between projects' overall cognitive level and complexity, suggesting that a larger project scope was not necessarily indicative of higher-order (or lower-order) cognition and vice versa (**Fig. S3** in Supplemental Materials).

Accessibility and inclusivity

The instructors of the course in 2020 (E.C.C. and K.M.C.) implemented intentional practices to ensure that the course was accessible for all students and that those with varying backgrounds and needs felt welcome and accommodated. Some practices were specific to the remote setting, while others are equally applicable to in-person teaching. Instructional approaches focused on active learning and student engagement can help to combat inequities in the classroom (Theobald et al., 2020), but equally important are strategies that promote a culture of respect and foster a sense of belonging for students (Dewsbury & Brame, 2019).

Virtual teaching – and adaptations such as virtual office hours – offered inherent accessibility benefits for students facing long commutes, disability-related accessibility challenges, and other barriers to attending classes on campus (Pichette et al., 2020). Virtual office hours offered added benefits for students who may perceive office hours as an unfamiliar, unsafe, or inaccessible space, with breakout rooms creating privacy for students with questions on assignments or personal matters. Students shared their enthusiasm for virtual office hours in final course evaluations (**Fig. 3**). Recorded lessons, the asynchronous Piazza Q&A board, a flexible attendance policy, and an option to submit a recorded final project presentation enabled the participation of students located in remote time zones due to the pandemic.

That said, virtual learning can make it harder to maintain focus and limit distractions. The large amount of screen time was the most frequently mentioned criticism in students' course evaluations (**Fig. 3**). "Zoom fatigue" is a form of exhaustion that may result from the intensity of continuous, close-up eye contact and seeing oneself, reduced mobility when having to stay in a video frame, and increased cognitive load from having to exaggerate nonverbal cues (Bailenson, 2021). As one student reported in their mid-quarter evaluation, "just being on Zoom for so long takes away my attention span." To mitigate these effects, regular breaks were taken during class, students were encouraged to take breaks during recorded videos, a video-optional policy was instituted on Zoom, and students were allowed to use the chat function to participate. Nonetheless, we acknowledge that teaching online to students with their cameras off can be disorienting. We remind prospective instructors teaching in a virtual setting for the first time to be kind to themselves.

In a survey distributed in the first week of class ("Assignment #0" in **Fig. 5a**), students were encouraged to introduce themselves to the teaching team by sharing their pronouns and any anticipated accessibility or learning needs. Survey responses helped instructors affirm students' identities and accommodate students' disabilities and led to instructors making an effort to accurately caption all lesson videos. The survey also asked about comfort with technology and prior exposure to coding, which we analyze in this study (as discussed in Methods). Previous coding experience was not required, and a prerequisite of one quarter of calculus from previous iterations of the course was removed. Instructors offered one-on-one mentoring as needed, recognizing that some students require additional, intensive help with certain topics or specialized guidance tailored to their specific learning style in order to keep pace with the class. These mentoring sessions also had the benefit of allowing those students to form a personal connection with the instructors, which is otherwise challenging in a large virtual classroom.

A classroom community built on safety and mutual understanding promotes engagement, especially among students with marginalized identities, by creating a supportive space to share ideas and ask questions (Barrett, 2021). In an online teaching environment, genuine care and a strong presence from instructors are particularly critical for creating student trust (Shay & Pohan, 2021) and keeping students engaged in learning (Prince et al., 2020). However, connection in the classroom can be difficult to promote in the absence of face-to-face instruction. With this in mind, community was intentionally fostered throughout the course. Community guidelines were co-created on the first day of class using an activity that asked both students and instructors to contribute their expectations of shared norms and endorse each other's contributions. At the start of each synchronous class, icebreaker activities asking students about their well-being and comfort with recent material primed them for participating. Warm-up activities like these have been shown to allay anxiety about classroom engagement, connect students with each other, and create a safer environment more conducive to active learning (Bledsoe & Baskin, 2014; Chlup & Collins, 2010). In general, the instructors cultivated connection by being easily accessible for questions, encouraging collaboration, and emphasizing that student physical and mental well-being were priorities throughout the course. In mid-quarter evaluations, one student noted that the "low stress environment" of the course helped them learn.

Course policies and expectations

Setting clear expectations supported by explicit guidance on how to succeed contributes to an accessible learning environment by establishing a safe and productive classroom culture and reducing confusion. The syllabus is the first opportunity to outline expectations. As such, a detailed course syllabus was drafted to include six student learning objectives (see Introduction), course and university policies, logistics, guidelines on Zoom etiquette, and a week-by-week schedule. Each of these components give students a clear understanding of what they should gain from the course, outline metrics for success, and create trust that the instructors have thoughtfully planned the curriculum (Habanek, 2005).

The syllabus also included an integrity policy that encouraged collaboration but prohibited plagiarism. Students were allowed to reference external resources such as online API documentation sites and Stack Overflow. Citations and acknowledgment of collaboration were expected in assignments, and students confirmed their agreement with the integrity policy in the initial survey (Assignment #0). In this way, the syllabus also acted as a contract that codified expectations for student behavior in the course (Eberly et al., 2001). No textbook was required in order to allow flexibility in the topics addressed and avoid high textbook costs that have a disproportionately negative impact on historically underserved students (Jenkins et al., 2020). That said, instructors could consider offering excerpts from textbooks as a supplementary resource. Some earth science-oriented Python textbooks now exist in print (e.g., Alyuruk, 2019; DeCaria & Petty, 2021; Esmaili, 2021) and online (Palomino et al., 2021; https://www.earthdatascience.org/courses/intro-to-earth-data-science/); a comprehensive text not specific to earth science is also freely available online (VanderPlas, 2016; https://jakevdp.github.io/PythonDataScienceHandbook/).

In-class participation and flipped video watching were not graded, partially in recognition of pandemic stressors but also to accommodate individual circumstances without requiring students to disclose possibly sensitive information. The expectation was that assignment grades would be sufficiently impacted if students were not engaged in these activities. For assignments that were graded, instructors offered a one-time, two-week extension to allow flexibility while still requiring students to learn foundational material. While lesson videos had high completion rates (**Fig. 5a**), implementing low-stakes graded comprehension checks could be useful in a situation of lower engagement (Jacobs et al., 2016).

Conclusions

Student experience

Overall, students perceived the course positively, rating its content, evaluation techniques, organization, and the course as a whole markedly higher than in past quarters (**Fig. 2**). These evaluations are notable given hardships related to the COVID-19 pandemic, as well as findings that show students often prefer passive lecturing over active learning due to the additional cognitive effort required to engage actively with material (Deslauriers et al., 2019). Students' view of the course content evolved from a critical stance expressed in mid-quarter evaluations, with comments citing its abstract or challenging nature, to an appreciative view of the data skills they had acquired by the end of the course (**Fig. 3**).

By calculating correlations between a variety of anonymized data sources (see Methods), presented in **Fig. 6**, we explore the impact of students' varying backgrounds and learning strategies on their course experiences and outcomes. We find that highly engaged students acquired more Python skills and earned higher grades. The correlation observed between three key metrics – Q&A forum days online, total lesson minutes watched, and number of forum answers – and the breadth of Python skills used in final projects suggests that highly-skilled students were more engaged with the course, acquired more content knowledge, and frequently shared that knowledge with peers. Variations in students' final Python skills cannot fully explain differences in their final grades, but the two show a positive nonlinear correlation. Students who earned higher grades tended to monitor the Q&A forum more frequently, collaborate more often with classmates, and watch lesson videos before class. A positive relationship between question-asking on a Q&A forum and final grades has been found in past work (Vellukunnel et al., 2017). Exposure to video content before working on related in-class activities may have helped students prepare for assignments, which comprised the majority of final grades. That said, the lack of correlation between Python skills used in final projects and the timing of video lesson views suggests that it was the total amount of time spent viewing lessons, not whether those lessons were watched before or after a class, that mattered most for students' application of course content to an open-ended project.

We find that students' self-assessment of programming skills in a final survey was not correlated with their final grades, consistent with research that found a weak correlation between tutor grades and self-assessments by over 3,000 undergraduate students (Lew et al., 2010). That said, students were asked to self-assess their Python competence, rather than their final grade, and the two metrics may not be entirely comparable. Nonetheless, this result could reflect the Dunning-Kruger effect, a cognitive bias in which those with the least knowledge tend to overestimate their performance or ability because they lack the competencies required for self-assessment (Kruger & Dunning, 1999). Students' final self-assessments were not correlated with any metrics other than prior coding experience, pointing to a persistent confidence from previous Python exposure that contributed to a perception of competence not necessarily reflected in grades or skills.

Significantly, neither students' final grades nor their code usage in final projects were correlated with prior coding experience, indicating that previous exposure to Python was not predictive of success in the course. That said, less prior experience was associated with higher engagement with lesson videos and the Q&A forum. This suggests a 'level playing field' in which those who came in with less previous knowledge of programming took full advantage of class resources to ultimately reach the same level of proficiency as their peers.

Recommendations for future teaching

We recommend without reservations adopting the key elements that we describe in this paper, particularly flipped instruction, an online coding platform and discussion board, and strong attention to accessibility. That said, we encourage others to improve on our framework and regularly seek feedback from students, preferably in a format that allows for anonymity. For example, in course evaluations, students encouraged the addition of more frequent, low-stakes practice of basic skills to reinforce fundamental concepts (see Course Elements section "Assignments"). New practice opportunities would ideally be coupled with immediate feedback that guides further practice, which promotes efficient learning and refinement of conceptual understanding (Ambrose et al., 2010). Additionally, data literacy skills could be taught through higher-level exercises asking students to scrutinize the limitations, biases, and provenance of scientific data sets and make predictions and recommendations grounded in their analysis of data (see, e.g., Kastens & Krumhansl, 2017). Instructors may consider expanding this offering into a multi-course sequence to incorporate these elements.

We acknowledge the ongoing paradigm shift in many scientific fields towards "open science," a broadly defined set of ethics that encapsulates practices like code reproducibility, curation of data for reuse, and open journal access (Brett et al., 2020; Ramachandran et al., 2021). While these practices were not explicitly taught in this course, its emphasis on collaborative programming, well-documented code, and the scientific method as an open, transparent endeavor speak to fundamental open science principles. Explicit instruction on advanced topics like reproducibility, data archival, version control using Git and GitHub (e.g., Blischak et al., 2016), manipulation of large data sets stored on the cloud (e.g., Gentemann et al., 2021), and the UNIX command line may be more appropriate for a separate, higher-level course.

The pandemic likely accelerated existing trends in higher education towards multi-modal instruction and more engaging teaching practices (Lockee, 2021). As universities have transitioned back to in-person teaching, we believe that the framework developed for this course is well-suited to a hybrid approach with in-person tutorial and work sessions but recorded lesson videos, opportunities for regular online engagement, and virtual office hours for accessibility. Alternatively, a fully remote version like that described in this study could still be offered, potentially with minimal penalty in student performance and satisfaction compared to in-person instruction (Ghosh et al., 2022; Ramirez et al., 2022).

Impact

OCEAN 215 recently became listed in the University of Washington's new cross-campus undergraduate Data Science Minor, which aims to bolster students' data literacy and programming skills within their field of study as well as other domains. The course has also had an impact outside of our university environment. The flipped lesson videos have been uploaded to a dedicated YouTube channel (https://www.youtube.com/@ocean215python), where they have been collectively viewed more than 13,000 times as of June 2023, reaching over 30 different countries.

Furthermore, the graduate student instructors have benefited from the professional development that teaching this course allowed. Opportunities such as this have been linked with the success of doctoral students earning their degree in a timely manner and attaining future employment in higher education (Bettinger et al., 2016). Our department plans for a rotating cast of two graduate students to continue serving as the primary teaching team, with the guidance and support of a dedicated teaching mentor to develop their pedagogical skills. Graduate students' ownership of the course will promote the teaching of current data science practices.

For many undergraduate students without a deeper interest in data science, however, multiple years may pass after completing OCEAN 215 before their next opportunity to use Python programming. For most, this comes in the form of their senior thesis. Students' demonstrated loss of coding skills during the intervening years (see Introduction section "Course history and development") suggests not only the merits of our improved instructional design but also an urgent need to infuse an oceanographic undergraduate curriculum with regular opportunities to practice and apply programming skills. Barriers to enacting this change include some instructors' lack of familiarity with Python – many, for example, use MATLAB for research – and the need to communicate a standard set of programming skills that students can be expected to know. In addition to infusing curricula with programming, effort could be invested in creating supervised research opportunities for students that involve the use of programming and data analysis skills. More broadly, we see the need for earth science undergraduate curricula to adopt active, student-centered pedagogical practices that more frequently allow students to construct knowledge through hands-on exploration of real-world data. Infusing earth science curricula with current data programming practices will naturally facilitate the achievement of these goals.

Data and code availability

The Python code used to generate the figures in this paper is available at https://github.com/ethan-campbell/Python_teaching_paper and archived on Zenodo (Campbell & Christensen, 2023). Anonymized class data are available by reasonable request from the corresponding author (E.C.C.).

Author contributions

E.C.C. and K.M.C. designed instructional materials, taught the course, conceived the study, analyzed the data, and wrote the initial manuscript. M.N. supervised the course. S.C.R. established the original course and acquired

funding. A.A., O.B., J.L., R.M., and I.O. participated in the student focus group and/or provided testimonials detailing their course experience. All authors provided input to the final manuscript.

Acknowledgements

We thank all the undergraduate students who we taught in OCEAN 215 for their participation, patience, and feedback during the course.

Funding

This work was supported by the University of Washington's School of Oceanography (E.C.C. and K.M.C.), the US Department of Defense through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program (E.C.C.), the National Aeronautics and Space Administration through award #80NSSC19K1252 (K.M.C.), and the National Science Foundation's Global Ocean Biogeochemistry Array (GO-BGC) Project under awards #1946578 and #2110258 (E.C.C. and K.M.C.).

References

Alammary, A. (2019). Blended learning models for introductory programming courses: A systematic review. *PLoS ONE*, *14*(9), e0221765.

Alyuruk, H. (2019). *R and Python for oceanographers: A practical guide with applications*. Elsevier.

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). What kinds of practice and feedback enhance learning? In *How learning works: Seven research-based principles for smart teaching* (pp. 121–152). John Wiley & Sons, Inc.

Anderson, R. E., Ernst, M. D., Ordóñez, R., Pham, P., & Tribelhorn, B. (2015). A data programming CS1 course. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 150–155.

Arms, S., Chastang, J., Grover, M., Thielen, J., Wilson, M., & Dirks, D. (2020). Introducing students to scientific Python for atmospheric science. *Bulletin of the American Meteorological Society*, *101*(9), E1492–E1496.

Ayer, V. M., Miguez, S., & Toby, B. H. (2014). Why scientists should learn to program in Python. *Powder Diffraction*, *29*(S2), S48–S64.

Bada, S. O. (2015). Constructivism learning theory: A paradigm for teaching and learning. *Journal of Research & Method in Education*, *5*(6), 66–70.

Bailenson, J. N. (2021). Nonverbal overload: A theoretical argument for the causes of Zoom fatigue. *Technology, Mind, and Behavior*, *2*(1).

Banchi, H., & Bell, R. (2008). The many levels of inquiry. *Science and Children*, *46*(2), 26–29.

Barrett, S. E. (2021). Maintaining equitable and inclusive classroom communities online during the COVID-19 pandemic. *Journal of Teaching and Learning*, *15*(2), 102–116.

Bettinger, E. P., Long, B. T., & Taylor, E. S. (2016). When inputs are outputs: The case of graduate student instructors. *Economics of Education Review*, *52*, 63–76.

Bledsoe, T. S., & Baskin, J. J. (2014). Recognizing student fear: The elephant in the classroom. *College Teaching*, *62*(1), 32–41.

Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. *PLOS Computational Biology*, *12*(1), e1004668.

Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. Longmans.

Brett, A., Leape, J., Abbott, M., Sakaguchi, H., Cao, L., Chand, K., Golbuu, Y., Martin, T. J., Mayorga, J., & Myksvoll, M. S. (2020). Ocean data need a sea change to help navigate the warming world. *Nature*, *582*(7811), 181–183.

Brookhart, S. M., & Chen, F. (2015). The quality and effectiveness of descriptive rubrics. *Educational Review*, *67*(3), 343–368.

Campbell, E. C., & Christensen, K. M. (2023). *Analysis code for "Cracking the code: An evidence-based approach to teaching Python in an undergraduate earth science setting"* (Version 1.0-as-submitted) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.8087944

Chlup, D. T., & Collins, T. E. (2010). Breaking the ice: Using ice-breakers and re-energizers with adult learners. *Adult Learning*, *21*(3–4), 34–39.

DeCaria, A., & Petty, G. W. (2021). *Python programming and visualization for scientists* (2nd ed.). Sundog Publishing.

Deslauriers, L., McCarty, L. S., Miller, K., Callaghan, K., & Kestin, G. (2019). Measuring actual learning versus feeling of learning in response to being actively engaged in the classroom. *Proceedings of the National Academy of Sciences*, *116*(39), 19251–19257.

Dewsbury, B., & Brame, C. J. (2019). Inclusive teaching. *CBE—Life Sciences Education*, *18*(2), fe2.

Donham, C., Pohan, C., Menke, E., & Kranzfelder, P. (2022). Increasing student engagement through course attributes, community, and classroom technology: Lessons from the pandemic. *Journal of Microbiology & Biology Education*, *23*(1), 1–6.

Eberly, M. B., Newton, S. E., & Wiggins, R. A. (2001). The syllabus as a tool for student-centered learning. *The Journal of General Education*, *50*(1), 56–74.

Ellwein, A. L., Hartley, L. M., Donovan, S., & Billick, I. (2014). Using rich context and data exploration to improve engagement with climate data and data literacy: Bringing a field station into the college classroom. *Journal of Geoscience Education*, *62*(4), 578–586.

Esmaili, R. B. (2021). *Earth observation using Python: A practical programming guide* (Vol. 75). American Geophysical Union and Wiley.

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, *111*(23), 8410–8415.

Furman, T., & Moldwin, M. (2021). Higher education during the pandemic: Truths and takeaways. *Eos*, *102*(9), 17–19.

Gentemann, C. L., Holdgraf, C., Abernathey, R., Crichton, D., Colliander, J., Kearns, E. J., Panda, Y., & Signell, R. P. (2021). Science storms the cloud. *AGU Advances*, *2*(2), e2020AV000354.

Ghosh, S., Pulford, S., & Bloom, A. J. (2022). Remote learning slightly decreased student performance in an introductory undergraduate course on climate change. *Communications Earth & Environment*, *3*, 177.

Granger, B. E., & Pérez, F. (2021). Jupyter: Thinking and storytelling with code and data. *Computing in Science & Engineering*, *23*(2), 7–14.

Grapenthin, R. (2011). Computer programing for geosciences: Teach your students how to make tools. *Eos, Transactions American Geophysical Union*, *92*(50), 469–470.

Greengrove, C., Lichtenwalner, S., Palevsky, H., Pfeiffer-Herbert, A., Severmann, S., Soule, D., Murphy, S., Smith, L., & Yarincik, K. (2020). Using authentic data from NSF's Ocean Observatories Initiative in undergraduate teaching. *Oceanography*, *33*(1), 62–73.

Haak, D. C., HilleRisLambers, J., Pitre, E., & Freeman, S. (2011). Increased structure and active learning reduce the achievement gap in introductory biology. *Science*, *332*(6034), 1213–1216.

Habanek, D. V. (2005). An examination of the integrity of the syllabus. *College Teaching*, *53*(2), 62–64.

Hadjerrouit, S. (2008). Towards a blended learning model for teaching and learning computer programming: A case study. *Informatics in Education*, *7*(2), 181–210.

Haine, T. W. N., Gelderloos, R., Jimenez-Urias, M. A., Siddiqui, A. H., Lemson, G., Medvedev, D., Szalay, A., Abernathey, R. P., Almansi, M., & Hill, C. N. (2021). Is computational oceanography coming of age? *Bulletin of the American Meteorological Society*, *102*(8), E1481–E1493.

Hanks, B. (2005). Student performance in CS1 with distributed pair programming. *ACM SIGCSE Bulletin*, *37*(3), 316–320.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.

Hays, J. D., Pfirman, S., Blumenthal, B., Kastens, K. A., & Menke, W. (2000). Earth science instruction with digital data. *Computers & Geosciences*, *26*(6), 657–668.

Hepworth, K., Ivey, C. E., Canon, C., & Holmes, H. A. (2020). Embedding online, design-focused data visualization instruction in an upper-division undergraduate atmospheric science course. *Journal of Geoscience Education*, *68*(2), 168–183.

Hodges, C. B., Moore, S., Lockee, B. B., Trust, T., & Bond, A. A. (2020, March 27). The difference between emergency remote teaching and online learning. *Educause Review*. https://er.educause.edu/articles/2020/3/the-difference-between-emergency-remote-teaching-and-online-learning

Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, *5*(1), 10.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95.

Irving, D. (2019). Python for atmosphere and ocean scientists. *Journal of Open Source Education*, *2*(11), 37.

Jacobs, C. T., Gorman, G. J., Rees, H. E., & Craig, L. E. (2016). Experiences with efficient methodologies for teaching computer programming to geoscientists. *Journal of Geoscience Education*, *64*(3), 183–198.

Jenkins, J. J., Sánchez, L. A., Schraedley, M. A. K., Hannans, J., Navick, N., & Young, J. (2020). Textbook broke: Textbook affordability as a social justice issue. *Journal of Interactive Media in Education*, *2020*(1), 3.

Kastens, K. A., & Krumhansl, R. (2017). Identifying curriculum design patterns as a strategy for focusing geoscience education research: A proof of concept based on teaching and learning with geoscience data. *Journal of Geoscience Education*, *65*(4), 373–392.

Kastens, K. A., Krumhansl, R., & Baker, I. (2015). Thinking big. *The Science Teacher*, *82*(5), 25–31.

Kastens, K. A., Zrada, M., & Turrin, M. (2020). What kinds of questions do students ask while exploring data visualizations? *Journal of Geoscience Education*, *68*(3), 199–219.

Klug, J. L., Carey, C. C., Richardson, D. C., & Darner Gougis, R. (2017). Analysis of high-frequency and long-term data in undergraduate ecology classes improves quantitative literacy. *Ecosphere*, *8*(3), e01733.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice*, *41*(4), 212–219.

Kruger, J., & Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, *77*(6), 1121–1134.

Kuksenok, K., Aragon, C., Fogarty, J., Lee, C. P., & Neff, G. (2017). Deliberate individual change framework for understanding programming practices in four oceanography groups. *Computer Supported Cooperative Work*, *26*(4–6), 663–691.

Lasser, J., Manik, D., Silbersdorff, A., Säfken, B., & Kneib, T. (2021). Introductory data science across disciplines, using Python, case studies, and industry consulting projects. *Teaching Statistics*, *43*(S1), S190–S200.

Lew, M. D. N., Alwis, W. A. M., & Schmidt, H. G. (2010). Accuracy of students' self-assessment and their beliefs about its utility. *Assessment & Evaluation in Higher Education*, *35*(2), 135–156.

Lin, J. W.-B. (2012). Why Python is the next wave in earth sciences computing. *Bulletin of the American Meteorological Society*, *93*(12), 1823–1824.

Lockee, B. B. (2021). Online education in the post-COVID era. *Nature Electronics*, *4*(1), 5–6.

Lopatto, D. (2010). Undergraduate research as a high-impact student experience. *Peer Review*, *12*(2), 27–30.

McCallum, S., Schultz, J., Sellke, K., & Spartz, J. (2015). An examination of the flipped classroom approach on college student academic involvement. *International Journal of Teaching and Learning in Higher Education*, *27*(1), 42–55.

McConnell, D. A., Chapman, L., Czajka, C. D., Jones, J. P., Ryker, K. D., & Wiggen, J. (2017). Instructional utility and learning efficacy of common active learning strategies. *Journal of Geoscience Education*, *65*(4), 604–625.

McDonnell, J., Lichtenwalner, S., Glenn, S., Ferraro, C., Hunter-Thomson, K., & Hewlett, J. (2015). The challenges and opportunities of using data in teaching from the perspective of undergraduate oceanography professors. *Marine Technology Society Journal*, *49*(4), 76–85.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002). The effects of pair-programming on performance in an introductory programming course. *ACM SIGCSE Bulletin*, *34*(1), 38–42.

McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of the 9th Python in Science Conference*, 56–61.

Met Office. (2022). *Cartopy: a cartographic Python library with a Matplotlib interface* [Computer software]. https://scitools.org.uk/cartopy

Middendorf, J., & Kalish, A. (1996). The "change-up" in lectures. *The National Teaching and Learning Forum*, *5*(2), 1–5.

National Academies of Sciences, Engineering, and Medicine. (2018). *Open science by design: Realizing a vision* *for 21st century research*. Washington, D.C.: The National Academies Press. https://doi.org/10.17226/25116

Old, P. L. (2019). *Bridging the gap in oceanographic data science curriculum: prototyping experiential learning materials in Python* [Undergraduate thesis, University of Washington]. ResearchWorks Archive. https://hdl.handle.net/1773/45628

Palomino, J., Wasser, L., & Joseph, M. (2021). *Introduction to earth data science textbook* (Version 1.5). Earth Lab CU Boulder. https://doi.org/10.5281/zenodo.3382162

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin*, *39*(4), 204–223.

Perkel, J. M. (2021). Ten computer codes that transformed science. *Nature*, *589*(7842), 344–348.

Pichette, J., Brumwell, S., & Rizk, J. (2020). *Improving the accessibility of remote higher education: Lessons from the pandemic and recommendations*. Higher Education Quality Council of Ontario. https://heqco.ca/pub/improving-the-accessibility-of-remote-higher-education-lessons-from-the-pandemic-and-recommendations/

Prince, M. (2004). Does active learning work? A review of the research. *Journal of Engineering Education*, *93*(3), 223–231.

Prince, M., Felder, R., & Brent, R. (2020). Active student engagement in online STEM classes: Approaches and recommendations. *Advances in Engineering Education*, *8*(4), 1–25.

Ramachandran, R., Bugbee, K., & Murphy, K. (2021). From open data to open science. *Earth and Space Science*, *8*(5), e2020EA001562.

Ramirez, S., Teten, S., Mamo, M., Speth, C., Kettler, T., & Sindelar, M. (2022). Student perceptions and performance in a traditional, flipped classroom, and online introductory soil science course. *Journal of Geoscience Education*, *70*(1), 130–141.

Rowe, P. M., Fortmann, L., Guasco, T. L., Wright, A., Ryken, A., Sevier, E., Stokes, G., Mifflin, A., Wade, R., Cheng, H., Pfalzgraff, W., Beaudoin, J., Rajbhandari, I., Fox-Dobbs, K., & Neshyba, S. (2021). Integrating polar research into undergraduate curricula using computational guided inquiry. *Journal of Geoscience Education*, *69*(2), 178–191.

Scheick, J., Leong, W. J., Bisson, K., Arendt, A., Bhushan, S., Fair, Z., Hagen, N. R., Henderson, S., Knuth, F., Li, T., Liu, Z., Piunno, R., Ravinder, N., Setiawan, L. D., Sutterley, T., Swinski, J. P., & Anubhav. (2023). icepyx: querying, obtaining, analyzing, and manipulating ICESat-2 datasets. *Journal of Open Source Software*, *8*(84), 4912.

Shay, J. E., & Pohan, C. (2021). Resilient instructional strategies: Helping students cope and thrive in crisis. *Journal of Microbiology & Biology Education*, *22*(1), 1–8.

Srinath, K. R. (2017). Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), 354–357.

Stains, M., Harshman, J., Barker, M. K., Chasteen, S. V., Cole, R., DeChenne-Peters, S. E., Eagan, M. K., Esson, J. M., Knight, J. K., Laski, F. A., Levis-Fitzgerald, M., Lee, C. J., Lo, S. M., McDonnell, L., McKay, T. A., Michelotti, N., Musgrove, A., Palmer, M. S., Plank, K. M., … Young, A. M. (2018). Anatomy of STEM teaching in North American universities. *Science*, *359*(6383), 1468–1470.

Theobald, E. J., Hill, M. J., Tran, E., Agrawal, S., Arroyo, E. N., Behling, S., Chambwe, N., Cintrón, D. L., Cooper, J. D., Dunster, G., Grummer, J. A., Hennessey, K., Hsiao, J., Iranon, N., Jones, L., Jordt, H., Keller, M., Lacey, M. E., Littlefield, C. E., … Freeman, S. (2020). Active learning narrows achievement gaps for underrepresented students in undergraduate science, technology, engineering, and math. *Proceedings of the National Academy of Sciences*, *117*(12), 6476–6483.

Thyng, K. M., Greene, C. A., Hetland, R. D., Zimmerle, H. M., & DiMarco, S. F. (2016). True colors of oceanography: Guidelines for effective and accurate colormap selection. *Oceanography*, *29*(3), 9–13.

VanderPlas, J. (2016). *Python data science handbook*. O'Reilly Media, Inc.

Vellukunnel, M., Buffum, P., Boyer, K. E., Forbes, J., Heckman, S., & Mayer-Patel, K. (2017). Deconstructing the discussion forum: Student questions and computer science learning. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 603–608.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Vázquez-Baeza, Y. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272.

Williams, L., & Upchurch, R. L. (2001). In support of student pair-programming. *ACM SIGCSE Bulletin*, *33*(1), 327–331.

Wilson, G. (2016). Software Carpentry: lessons learned. *F1000Research*, *3*, 62.

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P., & Wilson, P. (2014). Best practices for scientific computing. *PLoS Biology*, *12*(1), e1001745.

Wong, A. P. S., Wijffels, S. E., Riser, S. C., Pouliquen, S., Hosoda, S., Roemmich, D., Gilson, J., Johnson, G. C., Martini, K., Murphy, D. J., Scanderbeg, M., Bhaskar, T. V. S. U., Buck, J. J. H., Merceur, F., Carval, T., Maze, G., Cabanes, C., André, X., Poffa, N., … Park, H.-M. (2020). Argo data 1999–2019: Two million temperature-salinity profiles and subsurface velocity observations from a global array of profiling floats. *Frontiers in Marine Science*, *7*, 700.

Yuretich, R. F., Khan, S. A., Leckie, R. M., & Clement, J. J. (2001). Active-learning methods to improve student performance and scientific interest in a large introductory oceanography course. *Journal of Geoscience Education*, *49*(2), 111–119.

Figures

Figure 1. Key course elements: **(a)** Python platforms and software libraries that were taught (see **Table S4** in Supplemental Materials for specific functions, operators, and methods); **(b)** flipped video lessons, with a slide demonstrating how colors, fonts, design elements, and a minimal working example help to explain Python syntax; **(c)** class sessions focused on active learning, showing a completed portion of a group activity; **(d)** programming assignments, with an illustrative plot; **(e)** discussion on the Piazza Q&A forum, showing a student question and a peer answer endorsed by an instructor; **(f)** the final research project, represented as the sequence of assigned components; **(g)** underlying course elements that fostered an effective learning environment. Solid arrows indicate the progression from foundational material (a) to content delivery (b) and application (c); dashed arrows indicate the contributions of discussion forum engagement (e) to students' work on assignments (d) and the final project (f).

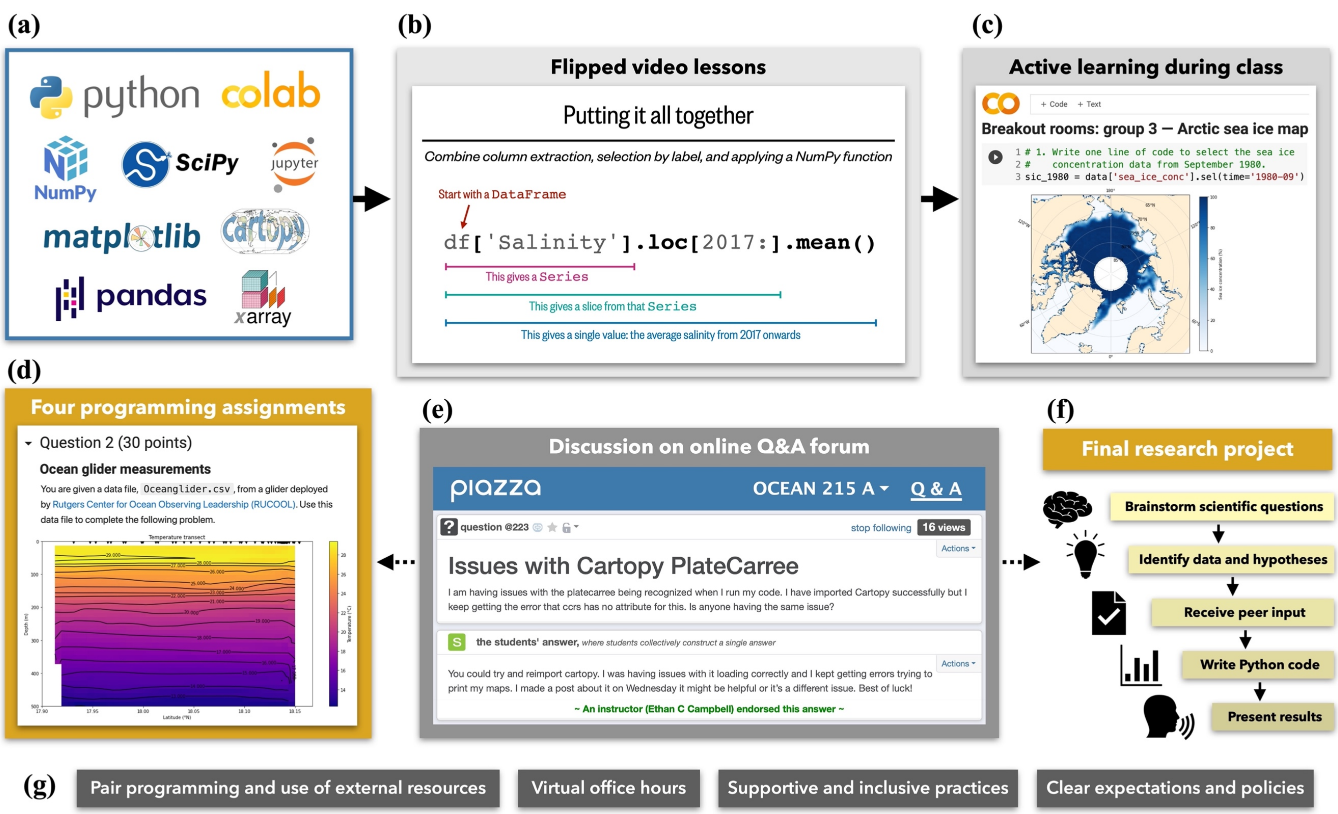


Figure 2. Selected metrics from anonymous end-of-quarter student evaluations in 2015, 2016, 2017, 2019, and 2020 (see Methods section "Initial, mid-quarter, and end-of-quarter surveys"). Differently worded questions were mapped between years as shown in **Table S2** in the Supplemental Materials. Metrics shown are class medians for 2015, 2016, 2017, and 2019 (gray crosses, except for "Total students enrolled"); 2015-2019 mean or 2020 class median (black points); and 2015-2019 standard deviation (bars). Note that y-axes have been truncated from the full 1-5 scale ("Very poor" to "Excellent") or 1-7 scale ("Much lower" to "Much higher"). For the full set of survey metrics, see **Fig. S1** in the Supplemental Materials.

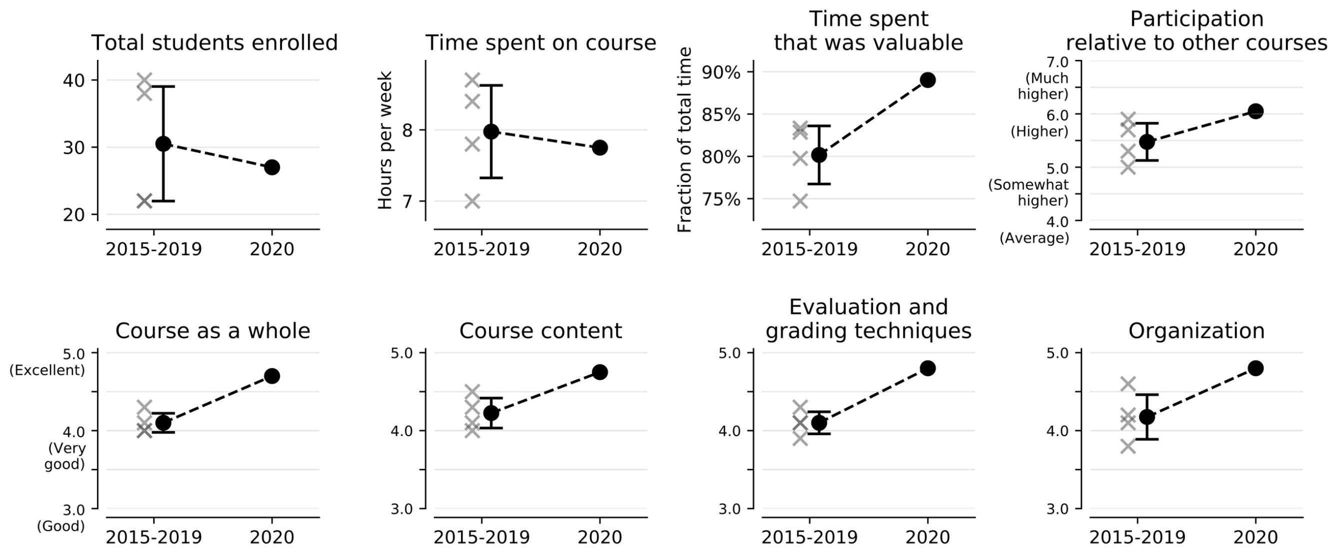


Figure 3. Themes identified in anonymous, open-ended student responses to mid-quarter (hatched bars) and end-of-quarter (solid bars) surveys in 2020, ranked according to the net positivity (blue) or negativity (red) of comments regarding those themes (see Methods section "Initial, mid-quarter, and end-of-quarter surveys"). Original survey prompts are listed in **Table S3** in the Supplemental Materials.

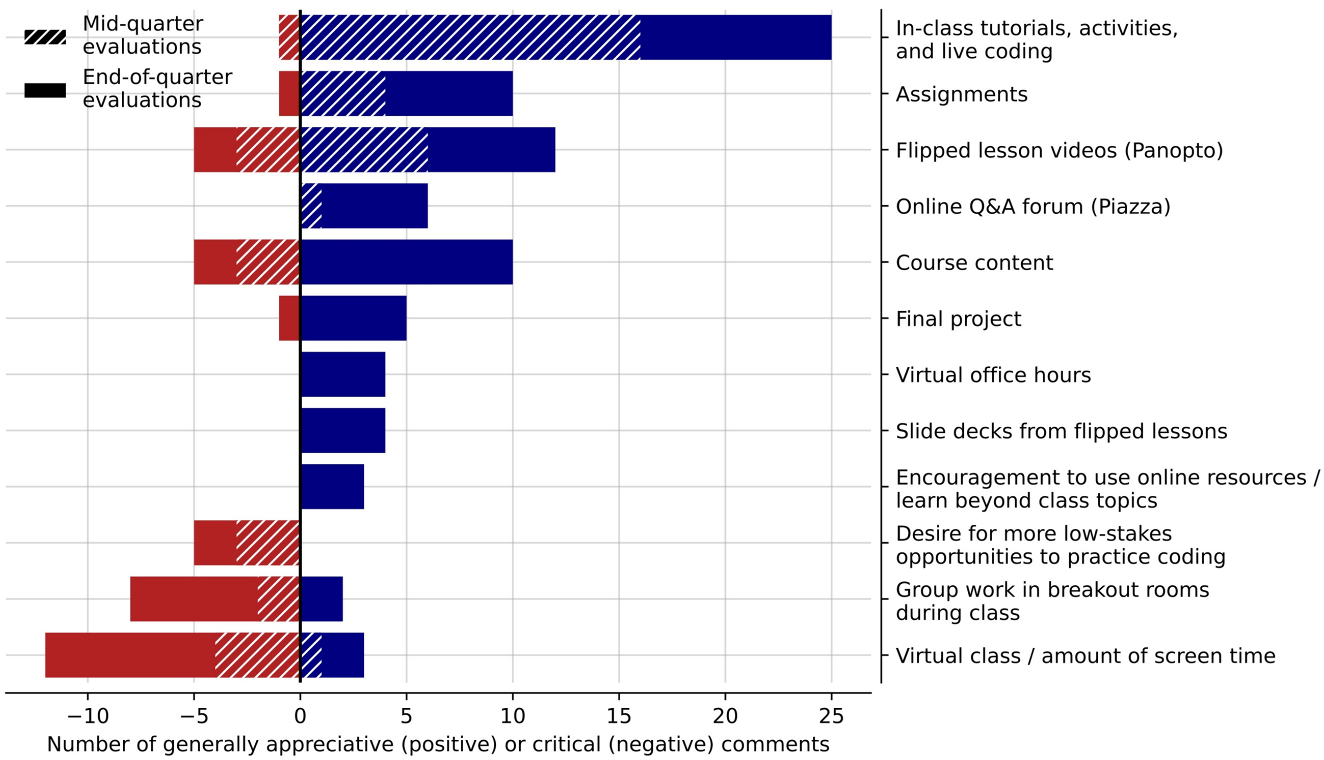


Figure 4. Timing of individual flipped (Panopto) video viewing sessions relative to the class for which each video was assigned. Overall watch timing is depicted as a filled histogram, similar to a cumulative distribution function, where each viewing session is weighted by its length, expressed as a fraction of the total video time delivered during the course (166.3 hours over $n = 41$ videos). The median and interquartile range (25%-75%) of video releases by instructors, relative to the corresponding class, is included for reference, indicating that videos were generally released 1.5 to 3 days before they were due. Note that vertical shading corresponds to days; also note the compressed positive x-axis scale.

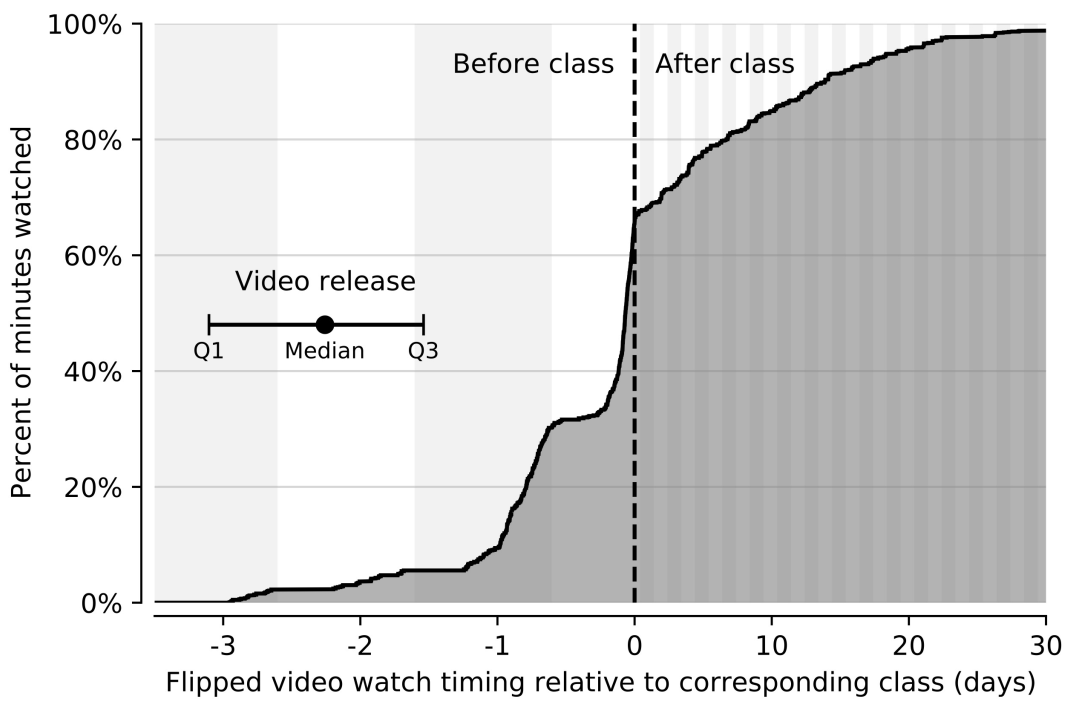


Figure 5. Student engagement with online platforms. **(a)** Flipped video completion rates (black lines) over time from Panopto are presented as both the class-wide median (dotted line) and average (solid line). Note that video completion by student was allowed to exceed 100% due to repeat views. Piazza Q&A forum engagement is shown as unique users per day (purple) and posts per day, segmented by the type of post (shaded colors; see legend). The timing of coursework deadlines (assignments ["A#..."] and final project checkpoints) are indicated with arrows. **(b)** Usage of the Piazza Q&A online forum by students and instructors, segmented by type of post (outer) and further divided by chosen audience (inner). "Required posts" were those requested from every student for Assignment #0 and final project check-ins. "Public posts" were viewable by all users, while "private posts" were visible to instructors only. "Anonymous posts" refer to those in which the author was hidden from other students, but not from instructors.

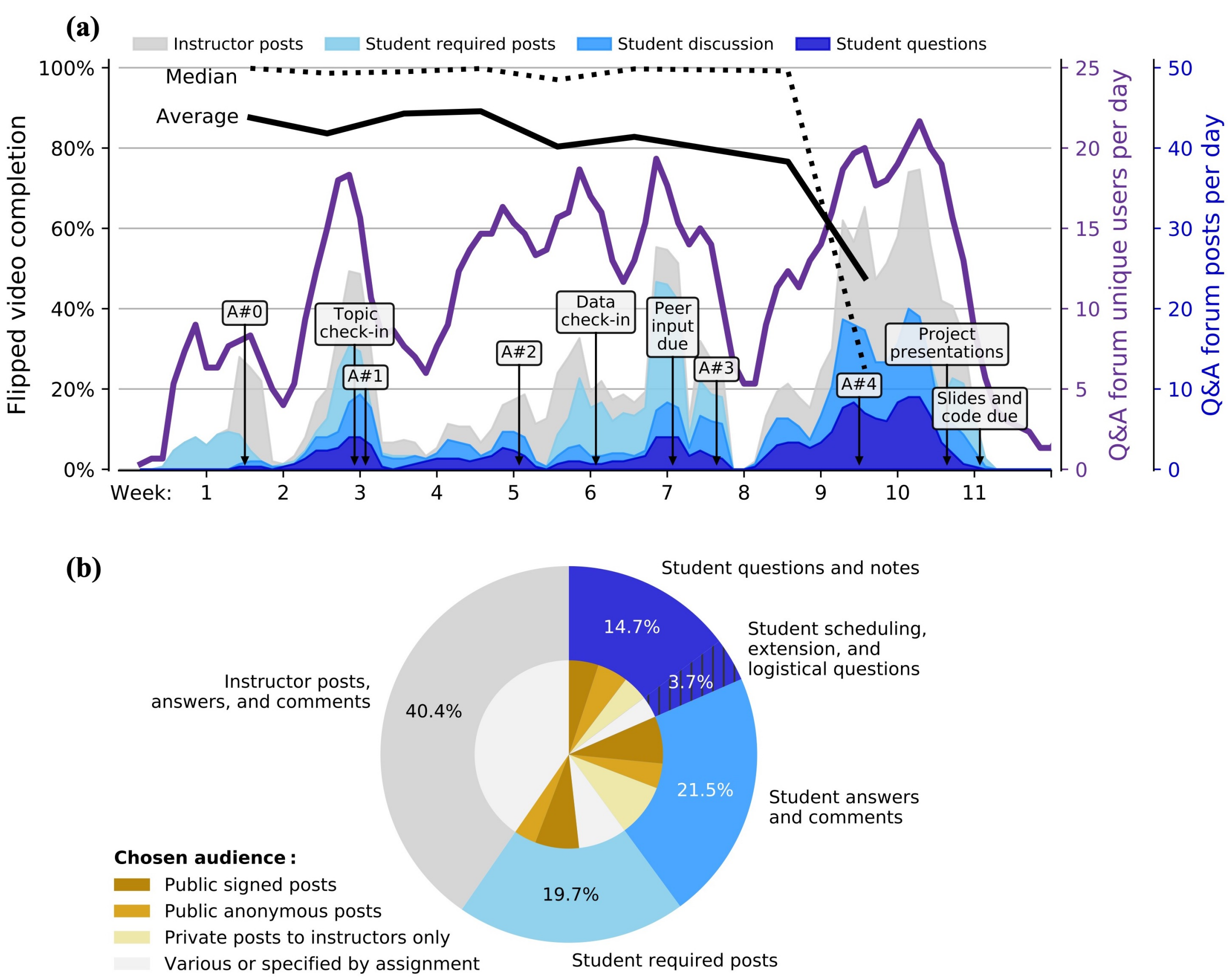
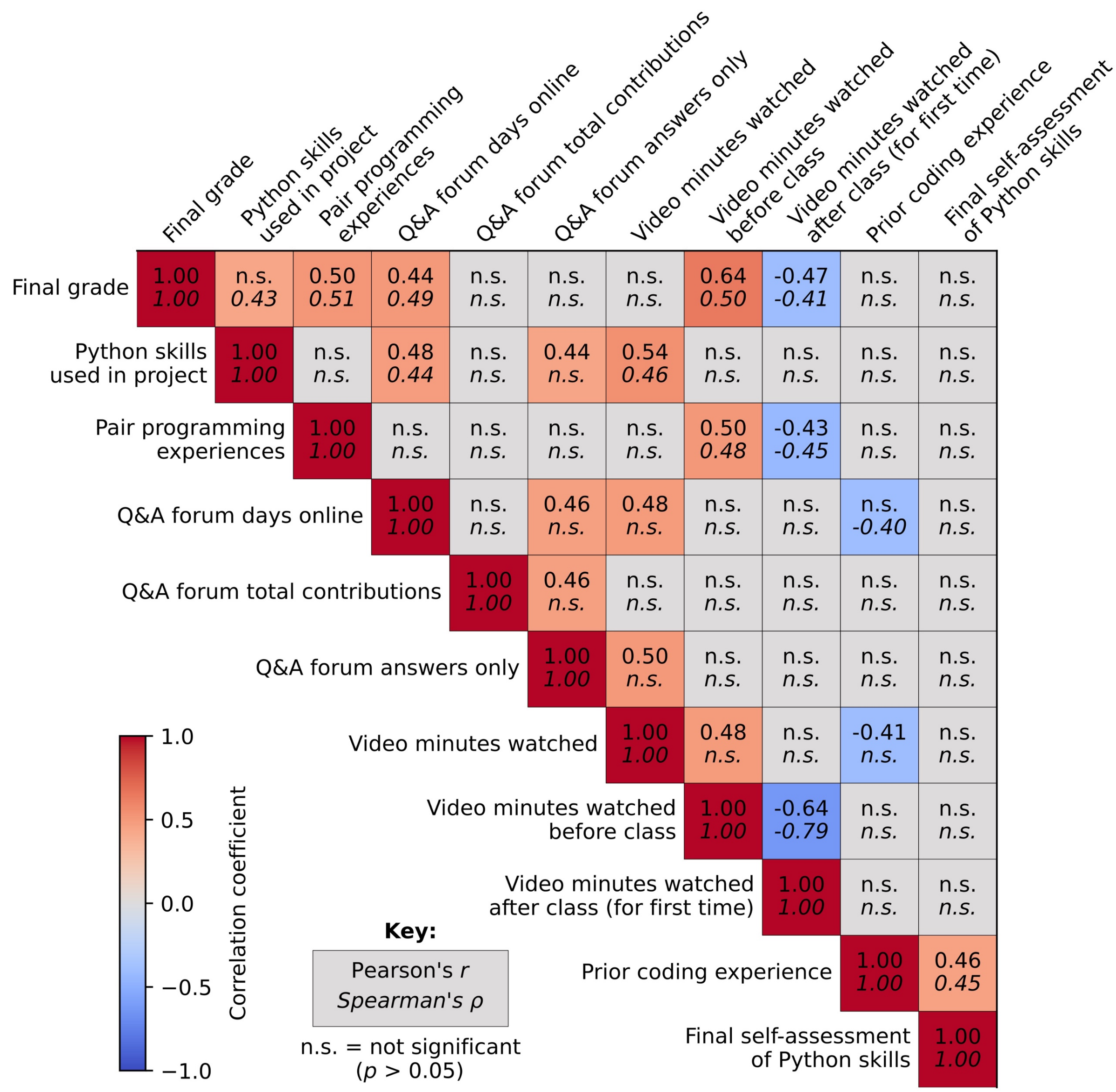


Figure 6. Correlations between student-specific anonymized metrics. Two tests were applied: Pearson's r (top values) and Spearman's ρ (lower values, italicized). Higher Pearson correlations indicate stronger positive linear relationships, while higher Spearman values indicate stronger monotonic relationships, which may not necessarily be linear. Correlations without statistical significance ($p > 0.05$) are indicated by "*n.s.*" For detailed information about the metrics presented, see Methods section "Final grades and programming skills" (for "Final grade"; column 1), **Table S4** in Supplemental Materials (for "Python skills used in project"; column 2), Course Elements section "Assignments" (for "Pair programming experiences; column 3), Methods section "Online forum engagement" (for Q&A forum-related metrics; columns 4-6), Methods section "Flipped video viewership" (for video-related metrics; columns 7-9), **Table S1** in Supplemental Materials (for "Prior coding experience"; column 10), and Methods section "Initial, mid-quarter and end-of-quarter surveys" (for "Final self-assessment of Python skills; column 11).



Tables

Table 1. Core topics and concepts taught in Ocean 215. Topics listed here are not necessarily in chronological order as taught in the course, and class time was not necessarily allocated in equal proportions to each topic.

Topic	Main concepts and skills
Why code in Python?	The power of programming is its versatility. Python is open source, stable, popular, free, and ideal for scientific data analysis. Google Colab offers advantages in a classroom setting compared to other programming environments.
Variables and object types	Variables store Python objects, which include numbers, booleans, strings, lists, tuples, dictionaries, and module-specific objects. Objects can be altered, indexed, sliced, iterated over, or used in mathematical operations. Assigning meaningful variable names makes for clearer code.
Logical operations and control flow	Objects can be compared using logical operations (and, or, is/equals, greater/less than, in, not). Loops and if-statements facilitate repetitive and conditional actions.
Packages and functions	Installing and using packages extends the capabilities of Python. Built-in, imported, and user-created functions accomplish common tasks and make for more compact, efficient code. Online documentation can be used to understand functions' arguments and outputs.
Data files	Oceanographic data are often stored in CSV and netCDF files, which can be read into Python, displayed, indexed, sliced, and manipulated using functions in the NumPy, Pandas, and Xarray packages. Real-world data sets can be obtained from public repositories and frequently contain messy or missing data.
Working with data	Data can be stored in multi-dimensional NumPy arrays and labeled structures specific to the Pandas and Xarray packages. These packages, as well as others like SciPy, have functions that average, sort, group, correlate, resample, smooth, regress, interpolate, and perform other computations on the data. Understanding common error types and tracing errors from their line of origin allows for methodical debugging of code.
Plotting	Line, scatter, bar, contour, pseudocolor, and other types of plots available from the Matplotlib package can be used to visualize data. Geospatial data can be projected onto maps using Cartopy. Appropriately customizing and labeling a plot is essential for interpretability.
Scientific skills	The modern scientific method is driven by data exploration, but also relies on traditional research skills like formulating hypotheses, interpreting the scientific significance of visualizations, effectively communicating results, and giving and receiving feedback from peers and mentors.

Table 2. Rubric used to classify students' final project questions and hypotheses based on the cognitive process dimension of the revised Bloom's taxonomy (Krathwohl, 2002). Higher levels of Bloom's taxonomy represent higher-order questioning and prediction. For the analyses in **Fig. S3** in the Supplemental Materials, multiple hypotheses and/or questions offered by students (up to three each) were assessed separately and weighted such that a student's three hypotheses, for example, would each contribute ⅓ of a point to their respective cognitive level's total count.

Cognitive level	**Questions**	**Hypotheses**
Level 3: Apply	*"What [happens if…]"* Intention to **execute** or **implement** a specific procedure, such as **calculating** a correlation; or *"Do [...]"* Intention to **answer** a binary (yes/no) question	**Specific results and relationships** (e.g., *the answer will be yes/no; X will show an increase over time; X and Y will show a positive correlation*)
Level 4: Analyze	*"How [does/do/is/are…]"* Intention to **characterize** or **test** a **straightforward** or **single-dimensional** relationship, phenomenon, or difference	**Contextual results and relationships** (e.g., *X and Y will show a positive correlation, but only under Z conditions; X and Y will vary with Z; X is characterized by Y patterns*)
Level 5: Evaluate	*"How [does/do…] affect…"* *"What [is/are…] the relationship between…"* Intention to **characterize** or **attribute** in an **open-ended** or **multidimensional** way; or *"Why [does/do/is/are…]"* Intention to **establish** causality by **integrating** external ideas or models and/or **connecting**, **contrasting**, or **weighing** multiple sources of information	**Explanations** (e.g., *X and Y will show a positive correlation because of mechanism Z; X and Y are different because of Z*)
Level 6: Create	*"What [does/do…] mean…"* *"How [does/do…] fit into…"* Intention to **evaluate** the **implications** of findings, **place** findings within old or new paradigms, **construct** or **produce** new frameworks, or **investigate** the **consequences** of phenomena using an open-ended approach	**Discovery** (e.g., *X is important because Y; X will differ from a past model Y, where a model is composed of two or more mechanisms; X can be explained using Y model;* or a hypothesis cannot be established due to lack of prior information)

Boxes

Box 1. Testimonials shared by undergraduate student coauthors (see Methods section "Student focus group" for more details). The students were encouraged to address one or more of the guiding questions listed in **Table S5** in the Supplemental Materials in their submissions.

———

Other coding classes that I have taken have generally failed to place skills in the context of applications. Without examples of methods being used, there is less of an incentive to understand them. In contrast, this course provided the opportunity to work with oceanographic data, allowing us to recognize the significance of the methods we were applying. For instance, ocean glider data was used to teach about interpolation. This was engaging because we first visualized the original, non-interpolated data and could see the gaps due to the physical motion of the device, then compared this with the data interpolated using the same axes and color scale.

Additionally, the lack of a textbook in this course made it easier to approach methods beyond what we learned in class. Instead, we learned to answer questions by accessing online resources like Stack Overflow. Doing so developed essential skills and gave me the confidence to apply new concepts in my final project. This meant my research could be dictated by my curiosity and questions, as it should be, and not by the limitations of what concepts we had covered in class.

In general, research can seem intimidating to many students because it relies on an individual's creativity. In other classes with exclusively rigid assignments and predetermined tasks, there is little opportunity for students to form original ideas, let alone develop them. In this class, we used creativity and critical thinking skills to develop a final project that answered an independently formed question. This experience has helped to prepare me for research. -O.B.

———

I previously took a Fortran class at the Ocean University of China, which had two traditional lectures and one lab each week. In that class, most students were not engaged during the lectures, which led them to be bewildered when doing real coding. I have also been teaching myself MATLAB for three years, basically learning by doing tasks with the help of the internet. This process has often been time-consuming, and it has been hard to organize my notes in a logical way. In comparison to those experiences, this course provided a logical pathway into Python, especially for oceanography applications. Without this class, it would have taken ten times longer to acquire the same knowledge, which would also have been less clear.

In class, Zoom breakout rooms forced everyone to discuss and practice the coding, which in turn forced us to come well-prepared for class. Though Google Colab has limited storage (RAM) and is unable to process large data sets, it is great for starters. Most of my other classes have been about theory and previously derived conclusions in the field, but this class has provided a bridge between theory and practice. After taking this course, I would say that we can now start to connect math and data to discover the areas of science we are interested in. -J.L.

———

I have always viewed research as something that is extraordinarily complicated. This class demonstrated that knowing a few basic Python functions and packages can provide a solid foundation to start conducting research. Additionally, offering this class as part of an oceanography curriculum instead of relying on a computer science department allowed us to learn about programming skills in a way that directly applied to our interests and studies.

I liked the way that the course was set up, in which we learned the material in an asynchronous video first and then practiced it in class. This helped me to discover where my gaps in understanding were and to learn from other people who may have understood a concept better than I did. Google Colab may not be the most powerful programming platform, but it is streamlined and easy to use, which made it great for first-time coders like me. Piazza was also an exceptionally useful resource.

Many classes present an idealized version of how research works. This class didn't. It was an important learning experience when my final research project didn't yield the correlation I expected. This was frustrating since I put so much time and effort into the project, but it showed that a lack of correlation can be an important result and that one's research doesn't always have to produce a major scientific breakthrough. -R.M.

———

I came in with a little prior coding experience thanks to robotic projects that I completed with my father as a child. In taking this class, the love of coding that I had as a child was reignited. I hadn't realized how beneficial and necessary knowing a programming language would be for research. Having Python in my arsenal opened up research opportunities that I wouldn't have been qualified for before and can aid me in branching out beyond oceanography in the future. The great experience I had in this class – and my realization that research and coding are extremely integrated – inspired me to pursue a minor in Data Science.

In this class, the coding assignments were based on real-world problem solving. I loved having the opportunity to work with a partner because we coded in completely different ways, and it was fascinating to see those differences. We were more effective together because we learned to compromise and collaborate to find the cleanest and fastest method between the two of us. Writing code on Zoom was a good alternative to in-person collaboration because we could share our screens and help pinpoint issues in each other's code. In addition, Piazza was helpful for me because it allowed anonymous or private questions, which avoids the uncomfortable feeling of asking a question that you think might be silly. I liked that we were able to get quick and helpful feedback on our code. It was a better way of communicating than those I have used in other classes, like email, which might get drowned out in a teacher's inbox, or Slack, which doesn't provide the anonymity that Piazza does. -I.O.

———