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 discipline-specific instructional approaches have been documented and assessed for their effectiveness in equipping geoscience undergraduate students with coding 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, synchronous 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 data from surveys, online learning platforms, student work, assessments, and a focus group, we conclude that the instructional design facilitated 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 higher engagement with course resources. We believe that the constructivist approach to teaching introductory programming and data literacy 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

[‡] Isabelle Osuna is now affiliated with School of Earth & Atmospheric Sciences, Georgia Institute of Technology, Atlanta, GA 30332, 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

Data programming has become the foundation of research in today's geoscientific disciplines. As the volume and size of 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 the 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, modeling and observation-focused oceanographers use highly unstandardized computational methods that may deviate from best practices in software engineering, as highlighted in an ethnography of oceanographers' programming practices (Kuksenok et al., 2017).

Discipline-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 may require 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 formal scientific computing instruction is often absent in earth science curricula, including oceanography (Old, 2019), except for highly scaffolded modules that employ coding in courses where programming is not the primary focus (e.g., Rowe et al., 2021). Even in courses that more extensively utilize programming within activity modules, such as those distributed by Project EDDIE (Environmental Data-Driven Inquiry and Exploration), pre-written code is usually provided to students (O'Reilly et al., 2022). 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 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 often involves active techniques such as project-based investigation, cooperative learning, and inquiry-based activities. These have been shown to improve student competencies in information recall, analysis, and quantitative reasoning in 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 students engaged in their learning due to the use of intentional teaching practices (Prince, 2004). Active modalities – including those designated as "high-impact educational practices" (Kuh et al., 2017) – 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). In a survey of almost 200 undergraduate oceanography professors, for example, three-quarters indicated that they use data in their instruction but are most likely to teach using lectures, rather than creating opportunities for active inquiry (McDonnell et al., 2015). There is strong evidence that using active learning techniques increases 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 supports the efficacy of active learning strategies in geoscience classrooms – particularly peer instruction, case studies, and problem-based activities (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 insufficient 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.

Our study reports on an evidence-based redesign of an undergraduate oceanography course that teaches introductory Python programming and 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.

Implementation

Course history and development

"Methods of oceanographic data analysis" (OCEAN 215) has been taught annually in the School of Oceanography at the University of Washington since its establishment in 2015. 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 later 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. Given that students had recognized the usefulness of the course content after completing the course (see **Fig. S1** in Supplemental Materials), we speculate that their subsequent hesitancy and lack of confidence in applying Python skills was due to a lack of recurrent exposure to Python in the curriculum (see Conclusions section "Impact" for more discussion) as well as weaknesses in the course design. Possible shortcomings include an overreliance on non-interactive lectures, a lack of student-driven inquiry, assignments' use of unrealistically clean scientific data, and course elements that were unnecessarily limiting or not reflective of current scientific Python practices.

The course was restructured (**Fig. 1**, **Table 1**) and subsequently co-taught during a 10-week quarter in 2020 by two graduate students, 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.

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 were isolated on campus or sequestered at home with family, mental health declined, and some became sick or had loved ones fall ill (Furman & Moldwin, 2021). With these realities in mind, the course redesign 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 pandemic, and a desire to infuse the course with active learning strategies. Changes included content that reflected the current scientific Python ecosystem (**Table 1**), cloud-based coding notebooks, flipped video lessons, discussions on an online question-and-answer (Q&A) forum, use of data from a wider range of earth science domains, an individually-driven final research project, encouragement of pair collaboration and use of external resources, and a syllabus with explicit policies, expectations, and the following end-of-quarter student learning objectives (SLOs):

1. Understand why the Python programming language is ideal for data analysis.
2. Write, execute, and debug Python code.
3. Access, read, transform, visualize, and interpret oceanographic data with confidence using Python.
4. Explore the ever-expanding universe of packages and tools available for creating and sharing code.

5. Formulate and investigate scientific research questions using programming and data analysis skills.
6. 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/.

Course content

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 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, while streamlining workflows starting from remote data access through to analysis and visualization (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. 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. The fact that Python is free prevents a reliance on expensive commercial solutions that can render analysis code inaccessible to scientists outside of well-resourced university environments (Gentemann et al., 2021). These qualities 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 conventional (quarter- or semester-long) course in the existing literature (Jacobs et al., 2016).

The updated OCEAN 215 covered 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 S1** 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 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 with 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.

Course elements

Programming platform

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, creating 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 can 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

A flipped classroom approach was implemented by assigning 14 recorded lessons of approximately 30 minutes each to be watched before synchronous (Zoom) classes. The 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 minutes during traditional lecturing (Middendorf & Kalish, 1996). In addition to segmenting videos, students were reminded to take breaks between segments. Flipped video watching and in-class participation 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.

Most lessons consisted of lectures 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.

Synchronous class sessions

In-class sessions were conducted using the Zoom platform. Each synchronous class started with simple icebreakers asking students about their well-being and anonymous polls to gather feedback about previous video lessons. Concepts from the relevant flipped videos were then briefly reviewed, leaving ample time for students to ask lingering questions. In some class sessions, short activities were used to introduce topics not covered in lesson videos.

The majority of synchronous class time was spent conducting live coding demonstrations and facilitating tutorials that integrated concepts taught in the videos. Compared to using slides or copying and pasting blocks of existing code, live coding 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). 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; (2) to build on concepts that students previously learned, encouraging reuse and modification of previous code; and (3) to compare and contrast different ways of achieving the same analytical or graphical result. Based on positive mid-quarter feedback, the instructors emphasized these tutorials and live coding in the second half of the course.

A Google Colab notebook was prepared for each class, presenting a tutorial with 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 4-5 person groups that worked cooperatively within Zoom breakout rooms. A "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 in their 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 provided guidance when needed. Groups' progress was tracked by watching a shared Google Doc configured ahead of time with templates in which each group filled in their final coding solutions. Occasionally randomizing group members allowed students to gain exposure to a variety of coding styles, social dynamics, and levels of confidence with the material.

Q&A forum

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**; note that alternative platforms with similar functionality exist, e.g., Ed Discussions). Piazza enables students to seek help on logistical or clarifying questions as well as their problem-solving processes, thereby reducing individual emails to instructors. The platform allows students to select the audience for their questions (instructors and/or classmates), post anonymously, respond to peers in threaded discussions, and collaboratively construct answers. Instructors may endorse and comment on student answers. 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 "Good work" or "I agree") was prescribed in the syllabus.

Assignments and final project

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 compared to class activities, allowing students more flexibility to design their own solutions. This created opportunities to practice both programming skills and data literacy, creating a foundation for more sophisticated independent analysis of data sets. Without a midterm exam, assignments were instructors' main window into student progress.

Students also completed an individually driven or collaborative final project (see **Text S1** in Supplemental Materials for the project description handout). The goal was for students to write code to answer a scientific question by exploring a data set of their choice, supported by ample guidance from the instructors and peer review from classmates. 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 classmates' choice of data or questions. The project culminated in the delivery of 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 S2** in Supplemental Materials). Rubrics may 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).

Students were offered the option to collaborate in pairs on both the assignments and final project. When programming as a pair, one student serves as the "driver," writing code, while the other observes, monitoring the code for defects and helping to problem-solve. Students were also allowed to reference external resources such as online documentation sites and Stack Overflow. Citations and acknowledgment of collaboration were expected in assignments and the final project, and students confirmed their agreement with the integrity policy in the initial survey (Assignment #0).

Evaluation

We adopt a two-pronged approach by first evaluating student achievement of SLOs using final project assessments, then exploring instructional approaches that helped students learn by using a variety of other data. The latter includes quantitative data from standardized course evaluations, an end-of-quarter student survey, engagement and usage metrics provided by the video and Q&A platforms, and graded assessments, along with qualitative data from the evaluations and student focus group. Prior to analysis, all student-specific metrics were de-identified and coded by a coauthor 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 coding in general, students were asked to share their prior experiences in an introductory survey distributed in the first week of class (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 S3** and **Fig. S3** 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. The introductory survey also encouraged students to introduce themselves to the teaching team by sharing their pronouns and any anticipated accessibility, technology, or learning needs.

We also obtained summary reports from end-of-quarter Instructional Assessment System (IAS) surveys completed by OCEAN 215 students in Autumn 2015, 2016, 2017, 2019, and 2020 (results from Spring 2015 and

Autumn 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 S4** 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. Changes between 2015-2019 and 2020 were tested for a statistically significant increase using a one-sided *t*-test for questions where increases could objectively be viewed as a desired improvement: metrics on a 1-5 ("Very poor" to "Excellent") scale and the metrics "Time spent that was valuable" and "Participation relative to other courses." Remaining metrics were tested for a statistically significant change in either direction using a two-sided *t*-test.

Furthermore, we apply a standard qualitative approach (Creswell, 1998) to extract meaning from students' anonymous responses to open-ended questions in 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 S5** in the Supplemental Materials. We identified common or unique themes mentioned by students, grouped similar themes, coded responses by noting whether a 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), and tabulated the frequency of each context for all themes (**Fig. 3)**. We also excerpt illustrative quotes from students' responses throughout the text.

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 course SLOs. The response rate was 92%. Submissions were not anonymous, but instructors guaranteed to students that their 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 video hosting and delivery platform used in the course, 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).

Q&A 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**.

Final projects

We use students' final projects as a barometer of their level of scientific reasoning, their final coding competency, and their achievement of course SLOs (**Fig. 7**). First, questions and hypotheses posed by students in their projects were assessed based on the seven levels of the cognitive process dimension of the revised Bloom's taxonomy (Bloom et al., 1956; Krathwohl, 2002; see rubric in **Table 2** for examples referenced in our classification), similar to the methodology of Kastens et al. (2020). Second, students' breadth of programming skills was evaluated computationally as the fraction of Python syntax elements taught in the course – namely, functions, operators, and methods – that were employed at least once in each student's submitted project code notebook (see **Table S1** in the Supplemental Materials for search terms used in the analysis). This metric varies widely between students (see Results section "Student learning outcomes") 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. Third, the submitted projects were graded using a rubric that was provided to students ahead of time to delineate expectations and evaluation techniques (**Table S2** in the Supplemental Materials). By mapping rubric subcategories onto four of the six corresponding SLOs (see Implementation section "Course history and development") and combining the graded scores within each category for each student, we create aggregate metrics of each student's final achievement of those key objectives.

Final grades

To represent overall student achievement, students' final grades are included in **Fig. 6** and **Fig. S3** in the Supplemental Materials. Grades were recalculated to ignore two students' incomplete assignments (0% grades) that occurred due to personal circumstances, 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%.

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 provide a short paragraph about course elements that affected their learning positively or negatively. Five students were chosen 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. Three focus group sessions were held in the quarter following Autumn 2020, each lasting 1-2 hours. In the sessions, the instructors asked questions designed to provoke open and candid discussion about students' perception of course elements and took notes by paraphrasing comments. Student participants did not have access to the anonymized student metrics described above.

The five students were additionally invited to share short testimonials detailing their unique experiences in the course and were offered coauthorship on the study (as noted below in Author Contributions). The four testimonials that were submitted are presented in **Text S2** in the Supplemental Materials and excerpted throughout the text. The final testimonials were assembled from students' responses to their selection of a subset of the guiding questions included as **Table S6** in the Supplemental Materials and were edited for style and grammar and to limit redundancy of themes mentioned. Insights gleaned from the focus group or testimonials 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.

Results

Student learning outcomes

Students' final project topics spanned the oceanographic, cryosphere, and atmospheric domains (**Fig. 7a**). Scientific questions and hypotheses posed by students largely map onto higher levels of Bloom's taxonomy, exemplifying higher-order questioning and prediction (**Fig. 7b, Fig. 7c**). The percentage of code syntax taught in the class that was used in each final project ranged widely from 6% to 29% (**Fig. 7d**) and exhibits no significant correlation with the assessed cognitive level of students' questions or hypotheses (not shown). In other words, students' level of scientific reasoning was not predictive of the analytical complexity of their finished projects. Overall final project grades were all above 80%, with most students scoring high marks (80% or above) on four project rubric categories representing the quality of their code, visualizations, use of data, and scientific research (**Fig. 7e, 7f, 7g, 7h**). These categories correspond to SLOs #2, #3, #5, and #6, with some overlap (see **Table S2** in Supplemental Materials). The widest spread in grades was in the category of scientific research (**Fig. 7h**), in which 28% of students scored below 80%.

By calculating correlations between a variety of anonymized data sources (see Evaluation), presented in **Fig. 6**, we explore the impact of students' varying backgrounds and learning strategies on their course experiences and outcomes. Significantly, neither students' final grades nor their code usage in final projects is correlated with prior coding experience, indicating that previous exposure to Python was not predictive of success in the course. Dichotomizing the class by prior coding experience (none/little versus some/moderate/lots) also reveals no statistically significant difference in final grades (**Fig. S3**). That said, less prior experience was associated with higher engagement with lesson videos and the Q&A forum (**Fig. 6**). Additionally, the positive correlation between three key metrics – total lesson minutes watched, number of Q&A forum answers, and forum days online – with the breadth of Python skills used in final projects indicates that students who demonstrated strong coding competency had likely acquired more content knowledge, frequently shared that knowledge with peers, and were more engaged with the course. Variations in students' demonstrated 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.

Role of course elements in student learning

<u>Course content</u>

Overall, students perceived the course positively, rating its content, evaluation techniques, organization, and the course as a whole markedly higher than in past years (**Fig. 2**). 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**). One focus group participant who was a first-time coder wrote in their testimonial (**Text S2** in Supplemental Materials):

> *"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."*

<u>Flipped structure</u>

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. While the total video lesson minutes watched by a student were correlated with the breadth of Python skills used in their final project, the timing of their video lesson views was not (**Fig. 6**).

Students in the focus group expressed that they appreciated the opportunity to watch videos at a convenient time and the ability to take breaks. 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. 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). 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.

Some students in the focus group reported re-watching videos to review material or using corresponding slide decks for the same purpose; one 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 estimated in final evaluations that the amount of time they spent each week was similar to past years. Yet out of students' total time spent on the course, nearly 90% was seen as valuable in advancing their education – a significant increase from past years ($p \leq 0.1$; **Fig. 2**).

<u>*Synchronous class sessions*</u>

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**). On the other hand, the large amount of screen time was the most frequently mentioned criticism in course evaluations (**Fig. 3**). Students also offered criticism on the use of breakout groups in their evaluations, 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." Several students wished for more time and instructor guidance in breakout rooms, which contributed to their overall negative rating (**Fig. 3**). Nonetheless, one focus group participant noted in their testimonial (**Text S2** in Supplemental Materials) that breakout rooms "forced us to come well-prepared for class" and in final course evaluations, students rated their overall participation as higher relative to other courses (6.0 on a 7-point scale, where 4.0 is "average"; **Fig. 2**).

Q&A forum

Students visited Piazza once every 1-5 days on average, and engagement 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. The forum saw 530 total student contributions, out of which two-thirds were voluntary, i.e., not required by a check-in or Assignment #0 (**Fig. 5b**).

Students selected the three audience options (public, either 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 overall 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."

Assignments and final project

In course evaluations, most students viewed the assignments and final project as beneficial (**Fig. 3**). Nearly half of the class – 48% of students – took advantage of the pair programming option at some point, with 34% of students collaborating on any given assignment or the project on average. Students generally chose the same classmate as their partner throughout the course. The number of times that a student worked collaboratively is presented as the metric "Pair programming experiences" in **Fig. 6**. One focus group participant shared their experience in their testimonial (**Text S2** in Supplemental Materials):

> *"… 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."*

The opportunity to synthesize course knowledge and the option to collaborate with classmates on final projects were specifically cited in students' evaluations as positive elements of the course. The ability to use external materials and learn beyond class topics was similarly welcomed (**Fig. 3**), and another student expressed in their testimonial:

> *"[Accessing online resources like StackOverflow] 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."*

That said, one critical survey comment related to ambiguity about the rigor of science expected and the open-ended nature of project checkpoints.

Discussion

Student learning outcomes

We measured students' achievement of key SLOs (#2, #3, #5, and #6) by assessing their final projects, with the assumption that the projects represent a holistic demonstration of students' capabilities. Those assessments indicated clear success in achieving learning objectives. Students produced impressive and original work that reflected earnest attempts to investigate scientific questions using effective coding and visualization techniques.

Consistent with research that found a weak correlation between tutor grades and self-assessments by over 3,000 undergraduate students (Lew et al., 2010), we saw no link between students' self-assessment of programming skills in a final survey and their final grades. A caveat is that students were asked to rate their Python competence, rather than their final grade, and the two metrics may not be entirely comparable. That said, this result could still 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). The lack of a relationship between students' final self-assessments and any metrics other than prior coding experience points to a persistent confidence from previous Python exposure that contributed to a perception of competence not necessarily reflected in higher grades or course-acquired skills. In contrast, our results suggest a "level playing field" in which those who came in with less previous knowledge of programming took full advantage of class resources, like lesson videos and Piazza, to ultimately reach the same level of proficiency as their peers, as shown in final grades and project code usage.

We believe the most novel aspect of this course was neither its content nor students' success at achieving SLOs but rather how the course was taught. An effective learning environment was intentionally created using evidence-based pedagogical elements: a mix of flipped lectures and engaging activities, a student-designed research project, opportunities for student collaboration, an online discussion forum, and efforts to center accessibility and foster classroom community.

Role of course elements in student learning

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). Consistent with those findings, our analysis of video watch timing, student focus group feedback, and course evaluations shows that our flipped structure enabled a diversity of strategies for content acquisition. 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. Nonetheless, our correlation metrics suggest that the total amount of time spent viewing lessons, not whether those lessons were watched before or after a class, was most influential in students' application of course content within their final projects.

In line with prior research on students' perspective of the flipped model (McCallum et al., 2015), our course structure generally received student approval in course evaluations (**Fig. 3**). Students' overall positive evaluations of the course 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).

Synchronous class sessions

Course evaluations indicated that in-class activities and demonstrations were well-liked and engaging. However, the facilitation of breakout groups and large amount of screen time presented challenges for students and instructors and were met with critical reviews. Though breakout rooms can allow for more individualized attention, the instructors had difficulty with distributing their finite time across groups and eliciting participation. Both can be linked to group size, and student focus group participants indeed shared mixed views on the number of students per group. Smaller groups could have encouraged more individual accountability at the expense of increasing demands on instructors' time as they cycle between breakout rooms. Larger groups would have enabled instructors to provide more efficient guidance and increased opportunities for peer instruction but often suffer from uneven participation. The optimal configuration may depend on individual classroom circumstances.

Q&A forum

The wide range of question types that we observe on Piazza are in line with previous research in an undergraduate computer science setting, which similarly showed high participation rates when students are encouraged to use the platform by teaching staff (Vellukunnel et al., 2017). Our correlation analysis of student metrics also matches the positive relationship between question-asking on a Q&A forum and final grades found in that prior study. The apparent efficacy of Piazza may reside in the fact that voluntarily 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 (Vellukunnel et al., 2017). Active learning would presumably be maximized if students use Piazza to seek help after they have invested time into trying different solutions and consulting other resources, which is encouraged by the asynchronous nature of the forum. 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. However, 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.

<u>*Assignments and final project*</u>

In each assignment notebook, copious scaffolding around each problem (e.g., step-by-step instructions, expected intermediate results, and links to documentation websites) was 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." The assignments were designed to be challenging yet were viewed favorably by both the student focus group and the final evaluation respondents. Both, however, indicated a desire for more short, frequent, low-stakes practice opportunities to help reinforce concepts and check understanding.

In contrast to instructor-generated activities, the final project allowed for student-designed questions and procedures. This encouraged "open inquiry," 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), students answered complex or potentially ill-structured questions using messy and incomplete real-world datasets (e.g., Ellwein et al., 2014; Klug et al., 2017) with instructor guidance mostly 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).

Pair programming has 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). 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. Nonetheless, 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 student pairs (Hanks, 2005).

Accessibility and inclusivity

Efforts were made to ensure that the course was accessible for all students and that those with varying backgrounds and needs felt welcome and accommodated. 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). A classroom community built on mutual understanding and respect promotes engagement, especially among students with marginalized identities, by creating a supportive space to share ideas and ask questions (Barrett, 2021).

The introductory survey helped the instructors affirm students' identities and accommodate disabilities, and students' responses led to instructors making an effort to accurately caption all lesson videos. While no students noted in the survey that they lacked computer or internet access, we shared relevant resources (e.g., a campus service for computer rentals and a public library program that loans internet hotspots) in the syllabus and initial class session.

Admittedly, 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. Warm-up activities like those we used at the start of synchronous classes 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 the context of a pandemic that saw many undergraduate students isolated from friends and support networks, the instructors cultivated connection and community by emphasizing that student physical and mental well-being were priorities throughout the course, encouraging collaboration, and being easily accessible for questions, including through Piazza. For assignments that were graded, instructors offered a one-time, two-week extension to allow flexibility while still requiring students to learn foundational material. In mid-quarter evaluations, one student noted that the "low stress environment" of the course helped them learn.

To ensure the broadest possible audience for the course, 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. This tutoring was also provided for students located in remote time zones in lieu of class sessions, among other accommodations. Individualized mentoring sessions have the benefit of allowing students to form a personal connection with the instructors, which is otherwise challenging in a large virtual classroom.

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). 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/).

Our overall approach of providing multiple modalities for student learning was consistent with a universal design for learning (UDL) framework that prioritizes equitable and inclusive teaching (Capp, 2017; Meyer et al., 2014). UDL outlines three core principles: (1) multiple means of representation, which our course accomplished through recorded lessons with text, auditory, and visual components, live coding demonstrations, and permissive use of external resources; (2) multiple means of action and expression, facilitated through practice opportunities and assignments with varying degrees of structure; and (3) multiple means of engagement, enabled by our use of individual as well as group work, verbal as well as chat-based participation, peer instruction, office hours, and the online forum.

Virtual teaching, including adaptations such as virtual office hours, offers 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 – regarded positively by students in course evaluations – offered added benefits for students who may have perceived office hours as an unfamiliar, unsafe, or inaccessible space, with breakout rooms creating privacy for students with questions on assignments or personal matters. Recorded lessons, the asynchronous 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.

That said, virtual learning can make it harder to maintain focus and limit distractions. "Zoom fatigue" is a particular 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). 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, though students' criticisms about screen time show Zoom fatigue remained a challenge. These solutions are also imperfect—breaks take class time, teaching to students with cameras off can be disorienting, and chat messages can be difficult to monitor during instruction.

Limitations

The robustness of our conclusions is limited by the relatively small sample size (25 students) and the study's focus on a single academic quarter. Additionally, the original course was offered a total of six times prior to Autumn 2020, but we were not able to obtain IAS survey responses from Spring 2015 and Autumn 2018. As such, these data are not included in our longitudinal comparison to previous years' course evaluations. In this comparison, we also cannot disentangle the various influences of the COVID-19 pandemic on learning from the impact of the curriculum changes that we made. Furthermore, we cannot quantify the impact of the new teaching team's positionality as graduate students on students' impression of course quality. A previous study, for example, found that professors who were perceived as younger received higher evaluations than professors teaching identical content who were perceived as older (Arbuckle & Williams, 2003).

While a pre-quarter assessment of student coding competency and attitudes would have been an ideal way to assess student growth, such an assessment was not conducted as the study design was conceived after the course had concluded. Data on students' age, race, and ethnicity were not collected for similar reasons, so we were unable to explore relationships between demographic profiles and students' experiences or success in the course. Likewise, student achievement for two of the six course SLOs (#1 and #3) could not be explicitly measured using available data, although an assessment of final projects found that students successfully met the remaining four SLOs.

While a systematic approach is used to identify and tabulate themes in the survey responses (see Evaluation section), we do not apply the same technique to qualitative data from the student focus group or their testimonials. The small sample size (five students) and the non-representative nature of the group selected by instructors would limit the appropriateness and utility of such an approach. Furthermore, the focus group conversations were not open-ended, but rather guided by questions formulated by instructors after initial analyses of other data (e.g., survey results, student learning metrics, etc.). Focus group discussions were documented through paraphrased notes rather than an exact transcription, so direct quotes are not presented. Testimonials were edited by instructors (as described in the Evaluation section "Student focus group"), further restricting the possibility of a quantitative thematic analysis approach.

Conclusions

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 Discussion section "Assignments and final project"). 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). While we did not implement graded comprehension checks for videos, these could be useful in a situation of lower engagement (Jacobs et al., 2016). 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 our 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 command-line interfaces 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). Though universities have transitioned back to in-person teaching, an interested and highly-engaged instructor team could still offer a fully remote version of this course, potentially with minimal penalty in student performance and satisfaction compared to in-person instruction (Ghosh et al., 2022; Ramirez et al., 2022). We believe that the framework developed for this course is also well-suited to a hybrid approach that incorporates in-person tutorial and work sessions but retains the pedagogical and accessibility benefits of recorded lesson videos, virtual office hours, and platforms that enable regular online engagement. Since 2020, this course has been offered annually in-person at the University of Washington by other graduate instructor teams with a flipped structure and most of the key curriculum elements introduced in this study.

Impact

The impact of this course extends beyond the students who enrolled in Autumn 2020. The flipped lesson videos were uploaded to a dedicated YouTube channel (https://www.youtube.com/@ocean215python), where they have been collectively viewed more than 22,000 times as of January 2024, reaching over 30 different countries. Furthermore, the graduate student instructors have benefited from the professional experience of developing a curriculum and managing a classroom. Opportunities such as this have been linked with the success of doctoral students 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.

For many undergraduate students without a deeper interest in data science, multiple years may pass after completing OCEAN 215 before their next opportunity to use programming. For most, this comes in the form of their senior thesis. Students' demonstrated loss of coding skills during past intervening years (see Implementation section "Course history and development") suggests not only the importance of our improved instructional design but also an urgent need to infuse an oceanographic undergraduate curriculum with regular, scaffolded opportunities to practice and apply programming skills. Barriers to enacting this change include the challenge of coalescing around a primary language of instruction while realizing the benefits of exposing students to other languages – many instructors, for example, use MATLAB for research – and a lack of curriculum mapping to communicate a standard set of programming skills that students can be expected to know and apply in courses. 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, 2024). 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 in 2015 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 are grateful to all the undergraduate students who we taught in OCEAN 215 for their participation, patience, and feedback during the course. We wish to thank the two anonymous reviewers and the journal editors whose helpful comments improved the quality and clarity of this study.

Disclosure statement

The authors report that there are no competing interests to declare.

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.

Arbuckle, J., & Williams, B. D. (2003). Students' perceptions of expressiveness: age and gender effects on teacher evaluations. *Sex Roles*, *49*(9–10), 507–516.

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. (2024). *Analysis code for "Cracking the code: An evidence-based approach to teaching Python in an undergraduate earth science setting"* [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.8087943

Capp, M. J. (2017). The effectiveness of universal design for learning: a meta-analysis of literature between 2013 and 2016. *International Journal of Inclusive Education*, *21*(8), 791–807.

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.

Creswell, J. W. (1998). Data analysis and representation. In *Qualitative Inquiry and Research Design: Choosing Among Five Traditions* (1st ed., pp. 139–165). SAGE Publications.

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.

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.

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.

Kuh, G., O'Donnell, K., & Schneider, C. G. (2017). HIPs at ten. *Change: The Magazine of Higher Learning*, *49*(5), 8–16.

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

Meyer, A., Rose, D. H., & Gordon, D. (2014). *Universal design for learning: theory and practice*. CAST.

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

O'Reilly, C. M., Josek, T., Darner, R. D., & Fortner, S. K. (2022). Pedagogy of teaching with large datasets: Designing and implementing effective data-based activities. *Biochemistry and Molecular Biology Education*, *50*(5), 466–472.

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 S1** 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).

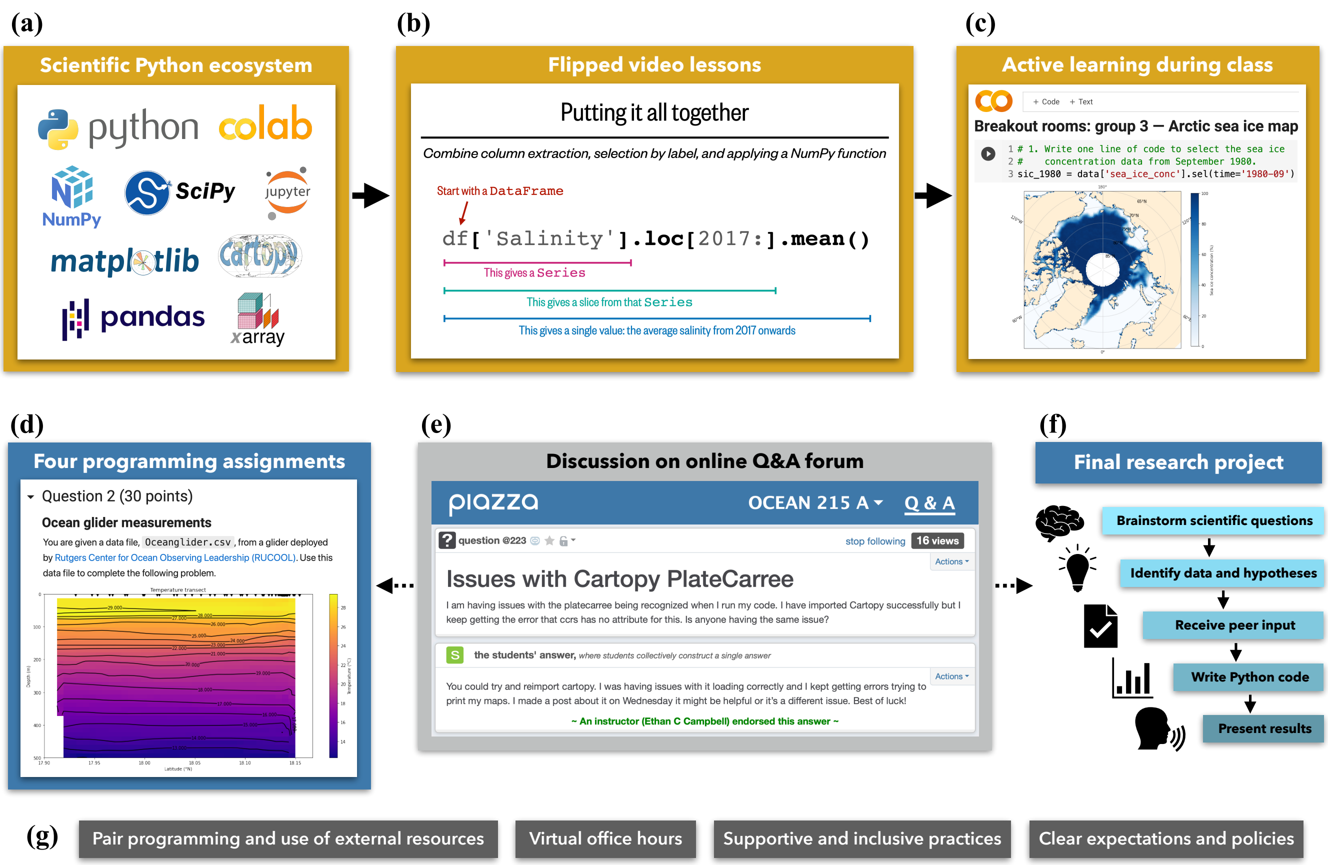


Figure 2. Selected metrics from anonymous end-of-quarter student evaluations in 2015, 2016, 2017, 2019, and 2020 (see Evaluation section "Initial, mid-quarter, and end-of-quarter surveys"). Differently worded questions were mapped between years as shown in **Table S4** in the Supplemental Materials. 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). Changes from 2015-2019 to 2020 were tested for a significant increase at the 95% (solid line) or 90% (dashed line) confidence level using a one-tailed *t*-test for all metrics except for "Total students enrolled" and "Time spent on course," which were tested for a significant change using a two-tailed *t*-test (no significant change was identified for either). For more details, see Evaluation section "Initial, mid-quarter, and end-of-quarter surveys." An absence of a line connecting the 2015-2019 and 2020 data indicates no statistically significant improvement or difference. 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.

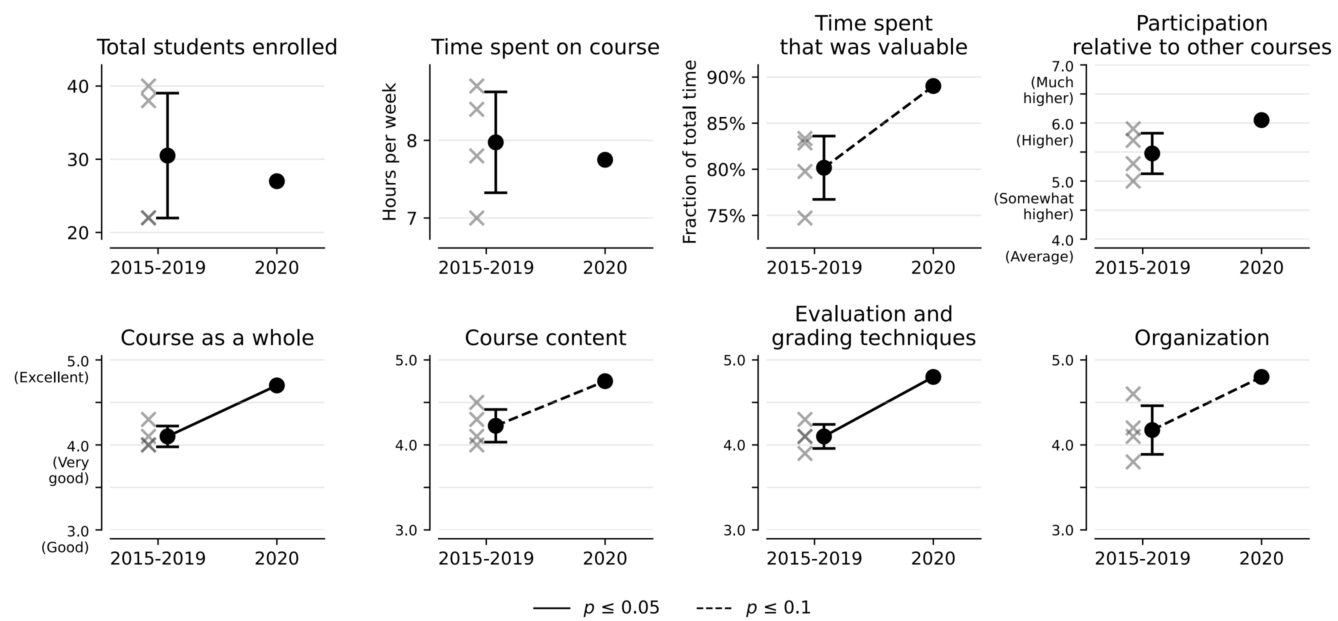


Figure 3. Themes identified in anonymous, open-ended student responses to mid-quarter (hatched bars) and end-of-quarter (solid bars) surveys in 2020, ranked according to the net positivity (blue) or negativity (red) of comments regarding those themes (see Evaluation section "Initial, mid-quarter, and end-of-quarter surveys"). Totals for mid-quarter and end-of-quarter evaluations are stacked, not overlapping, within each bar. Original survey prompts are listed in **Table S5** in the Supplemental Materials.

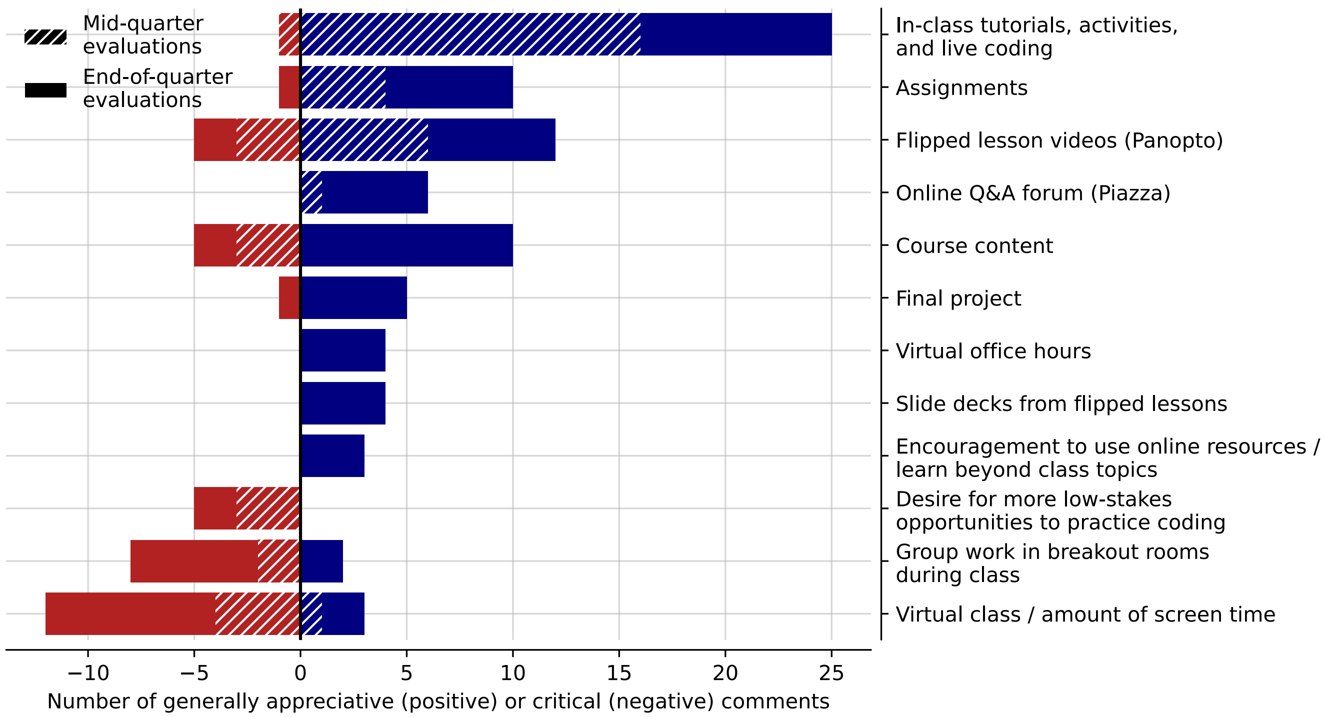


Figure 4. Timing of flipped Panopto video viewing sessions relative to the class for which each video was assigned. Viewing sessions were binned along the x-axis according to their timing before or after their corresponding class (with each viewing session weighted by its duration), then total minutes for each timing bin were summed from left to right to produce the cumulative distribution of watch timing shown here. The y-axis is the cumulative fraction of total video time delivered during the course (166.3 hours over 41 videos), with video rewatches included. 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. Vertical shading corresponds to days; note the compressed positive x-axis scale.

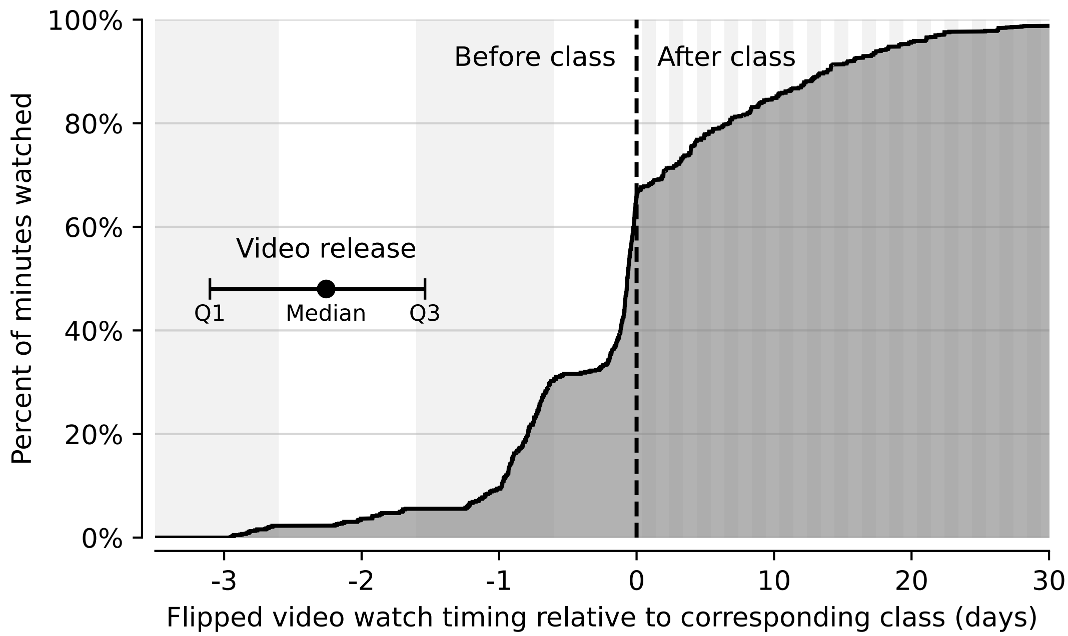


Figure 5. Student engagement with online platforms. **(a)** Flipped video completion rates over time from Panopto are presented as both the class-wide median (dotted black line) and average (solid black line). Note that video completion by student was allowed to exceed 100% due to repeat views. Piazza Q&A forum engagement is shown as unique users per day (purple) and posts per day, segmented by the type of post (shaded curves; see colors in 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.

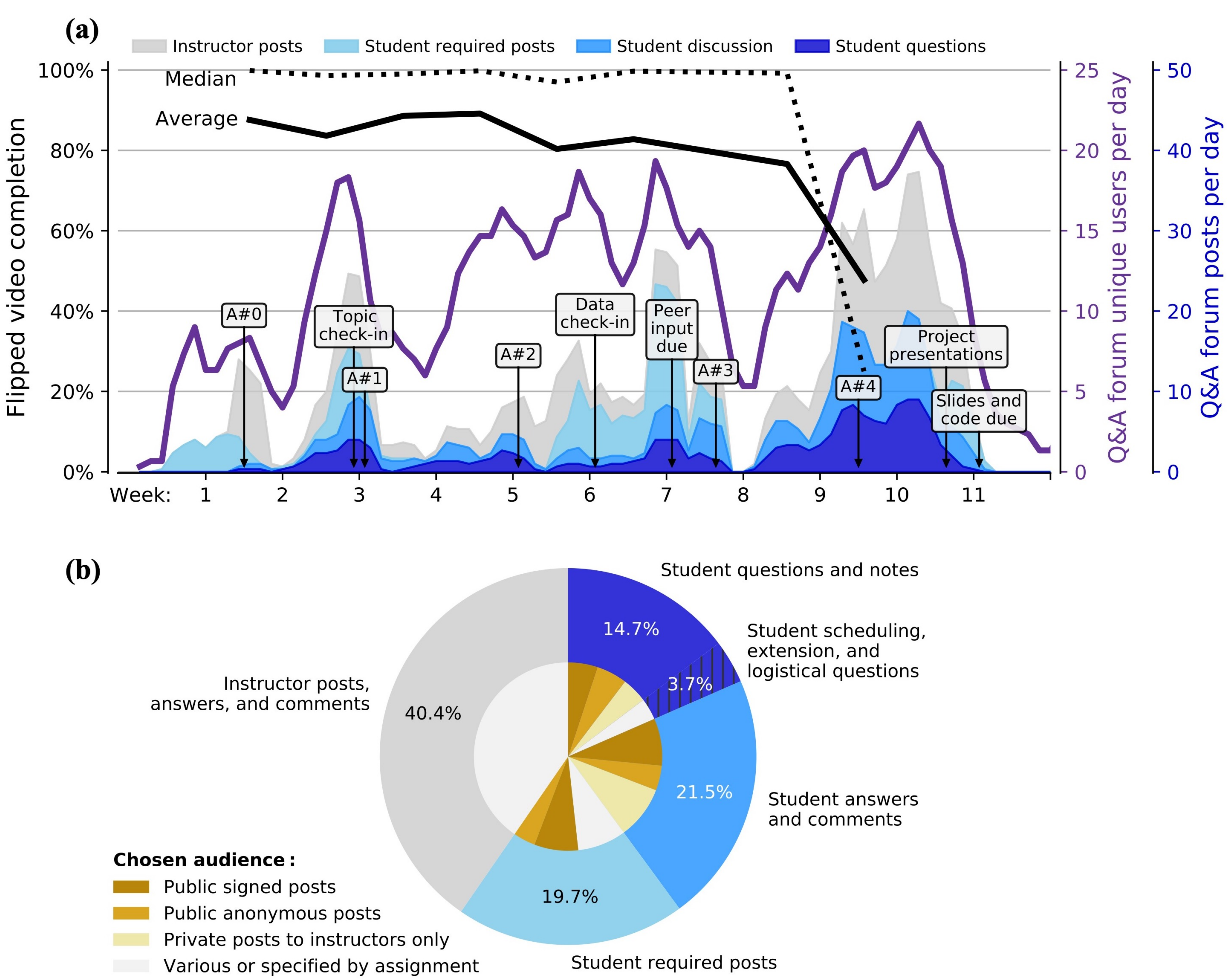


Figure 6. Correlations between student-specific anonymized metrics. Two tests were applied: Pearson's *r* (top values) and Spearman's *ρ* (lower values, italicized). Higher Pearson correlations indicate stronger positive linear relationships, while higher Spearman values indicate stronger monotonic relationships, which may not necessarily be linear. Correlations without statistical significance ($p > 0.05$) are indicated by "*n.s.*" Colors correspond to the larger of the two correlation coefficients by absolute value. For detailed information about each metric presented, see Evaluation section "Final grades" (for "Final grade"; column 1), **Table S1** in Supplemental Materials (for "Python skills used in project"; column 2), Results section "Assignments and final project" (for "Pair programming experiences; column 3), Evaluation section "Q&A forum engagement" (for Q&A forum-related metrics; columns 4-6), Evaluation section "Flipped video viewership" (for video-related metrics; columns 7-9), **Table S3** in Supplemental Materials (for "Prior coding experience"; column 10), and Evaluation section "Initial, mid-quarter and end-of-quarter surveys" (for "Final self-assessment of Python skills; column 11).

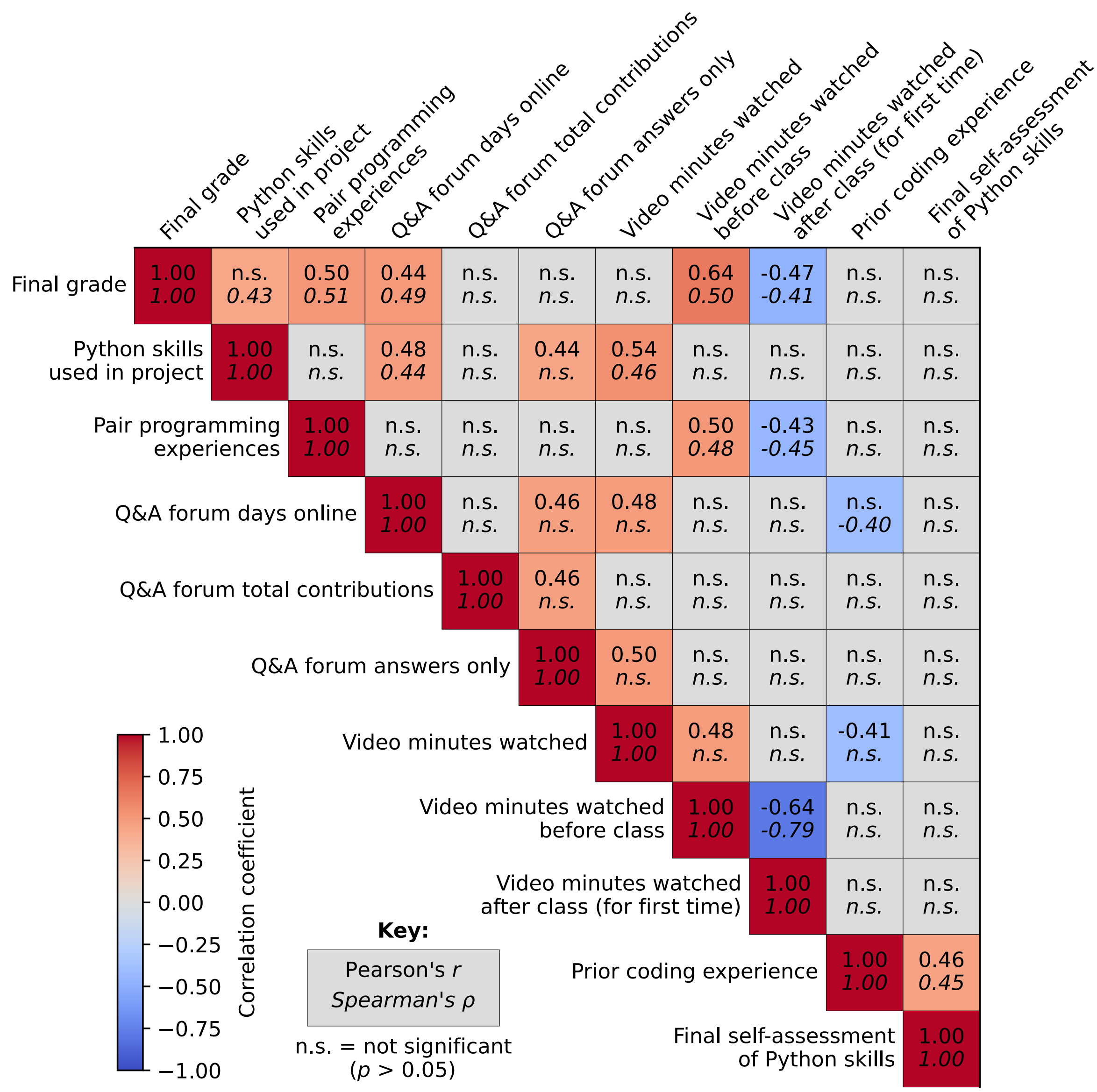
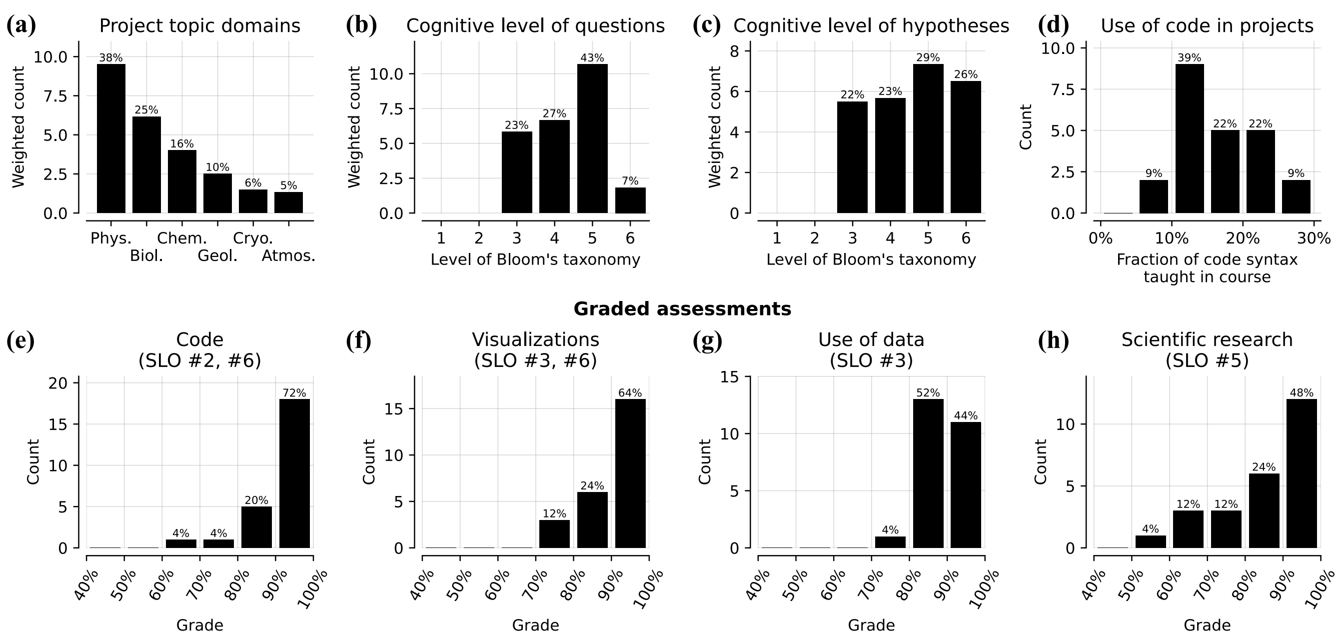


Figure 7. Assessment of students' final projects. **(a)** Distribution of domains of students' final projects. If a project topic touched multiple domains, each domain was weighted such that, for example, a project spanning three domains would contribute ⅓ of a point to each of the domains' total count. **(b-c)** Distribution of cognitive level of students' questions and hypotheses. Each student's questions and hypotheses (up to three each per student) were assessed based on the cognitive process dimension of the revised Bloom's taxonomy (Krathwohl, 2002) using the rubric and weighting described in **Table 2**, with higher levels of Bloom's taxonomy representing higher-order questioning and prediction. **(d)** The fraction of code syntax taught in the course that students used in their projects (see **Table S1** in Supplemental Materials for assessment methodology and search terms). **(e-h)** Project grades within four named categories that correspond to student learning objectives (SLOs) outlined in the text (see **Table S2** in Supplemental Materials for grading rubric and mapping to SLO categories). These categories (with rubric subcategories in parentheses) are code (correctness, functionality, tidiness, perseverance), visualizations (clarity, colormaps, labels, creativity), use of data (data collection, processing, results/interpretation), and scientific research (background, questions/hypotheses, explanations). Note the x-axes are truncated to 40%-100% for readability.



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	Key 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 allow 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). For the analyses in **Fig. 7b** and **Fig. 7c**, 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 1: Remember	*"Can the data be visualized using my skills?"* Intention to **recall** coding techniques taught in the course and **recognize** their proper use	**Recall of course material** (e.g., *the data can be depicted using a scatter/line/pseudocolor plot*)
Level 2: Understand	*"What stands out in the data?"* Intention to **summarize** the data; or *"Do the data resemble what we expect the ocean to look like?"* Intention to **interpret** the data and **classify** what is present by **comparison** to known examples	**Factual interpretation** (e.g., *the data will have X, Y features; the data will resemble X, Y other ocean data*)
Level 3: Apply	*"What [happens if…]"* Intention to **execute** or **implement** a specific procedure, such as **calculating** a correlation; or *"Does […]"* 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 exhibit different features 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)