From Zero to Python in 10.5 hours: Building Foundational Programming Skills with Marine Biology Graduate Students and Researchers in an Introductory Workshop Series

Geoffrey P. Timms, Marine Resources Library, College of Charleston, 66 George Street, Charleston, SC 29424, USA. timmsgp@cofc.edu. https://orcid.org/0000-0003-0970-2618

Jeffrey R. Guyon, National Oceanic and Atmospheric Administration, National Ocean Service, National Centers for Coastal Ocean Sciences, Hollings Marine Laboratory, Charleston, SC 29412, USA. jeff.guyon@noaa.gov https://orcid.org/0000-0003-3358-1318 https://loop.frontiersin.org/people/1770432/overview

Abstract

Programmatic processing, analysis, and visualization of scientific research data necessitate computational skills that scientists do not consistently acquire during their education. Python is a programming language currently in the ascendant among the scientific community. In early 2022, 25 marine scientists and marine biology graduate students were introduced to the fundamentals of Python programming in a 10.5-hour workshop series offered over seven weeks. A functional, web-based Python environment using Jupyter Notebooks, JupyterLab, Binder, and GitHub was paired with scaffolding and active learning pedagogy, to ensure an engaging learning experience. We describe the course design, adaptations, and outcomes, and offer recommendations for developing a similar introductory programming workshop.

Keywords: Python, Coding, Programming, Jupyter notebooks, Training, Workshop

Introduction

The evolution of science and technology has significantly increased the volume of data that researchers can generate and has diversified the methods by which data can be analyzed and visualized. Techniques to clean, synthesize, and analyze data have also evolved to harness ever expanding computational power (Lewis et al., 2018). From peptide-taxon attribution in marine metaproteomics (Saunders et al., 2020) to determining bioturbation levels by analyzing marine core images (Casanova-Arenillas et al., 2020), marine scientists increasingly programmatically process and analyze data. This requires that researchers acquire and maintain a computational skill set that enables them to efficiently accomplish a variety of data processing tasks throughout the research data lifecycle (Ekmekci et al., 2016). This paper introduces a workshop series developed to support marine biology graduate students and marine scientists in the development of foundational Python programming skills.

Python (https://python.org) is a programming language that is well suited to data processing in biological science, especially for both students and established researchers who have not previously used a programming language (Badenhorst et al., 2019; David, 2021). Python is also attractive due to the availability of many discipline-specific add-on packages such as SciPy (algorithms for scientific computing, https://scipy.org) and Biopython (computational molecular biology, https://biopython.org/). The original Python programming language was published in early 1991 and, over the last 31 years, over 25 version releases have been distributed. All Python releases are open source and available at https://www.python.org as no cost downloads available in Linux, macOS, and Windows distributions through the gracious support of numerous donors. Given its ease of use and powerful programming capabilities, Python is one of the most popular programming languages in use today supporting web

applications, scripting, database, data science and machine learning applications. Python now has over 10.1 million active developers; a number which has been increasing with Python's popularity in supporting data science applications used by an estimated 70% of machine learning developers (Voskoglou et al., 2021).

The College of Charleston is a public liberal arts institution, located in Charleston, South Carolina. While the main campus is located in downtown Charleston, the college's Grice Marine Laboratory and the associated Graduate Program in Marine Biology (GPMB) are located to the southeast of Charleston, across Charleston Harbor, on the historic Fort Johnson Campus. The lab supports teaching and research in evolutionary biology, marine biogeography, cellular and molecular biology, benthic ecology, immunology, microbial ecology, phytoplankton ecology, environmental physiology, fish systematics, invertebrate zoology, marine genomics, and other marine science disciplines (https://gricemarinelab.cofc.edu/). Students in the graduate program typically learn R, a programming language for statistical computing and visualization, to support their thesis research.

The GPMB is co-located with the South Carolina Department of Natural Resources Marine Resources Research Institute (MRRI) and the Hollings Marine Laboratory which is part of the National Oceanic and Atmospheric Administration's (NOAA) National Centers for Coastal Ocean Science (NCCOS) with research partners from the National Institutes of Standards and Technology (NIST) and the Medical University of South Carolina (MUSC). This research environment provides opportunities to connect the College of Charleston marine biology students and professors with state and federal coastal researchers, supporting a highly collaborative and unique location for conducting marine research into South Carolina's natural treasures. As with many other locations in the country, these collaborations were interrupted over

the last few years with several closures following the COVID epidemic in 2020 through 2022, heavily impacting student research and access for others working on the campus.

The Marine Resources Library is located inside the MRRI and is operated as a branch of the College of Charleston Libraries. It is staffed by one faculty librarian and one Library Technical Assistant. As a collaborative endeavor, both supported by and serving the partner institutions at Fort Johnson, the librarian has implemented initiatives to train its constituents in a variety of library-based skills. As a regular user of Python, the librarian sought to assess the level of interest among the Fort Johnson research community in learning this programming language, through a survey distributed on August 26, 2021. Participants selected a response on a five-point Likert-type scale to the question "How interested would you be in joining a Marine Resources Library workshop series to gain practical introductory-level experience with the Python programming language?" They were also asked to indicate their current level of experience with a programming language on a Likert-type scale and to provide their institutional affiliation.

A total of 76 responses (representing approximately 30% of the campus scientific workforce) was encouraging, especially given that most of the staff and students are not computer programmers, but more likely looking for tools to help them best analyze their data. The results from the survey showed that 53 of the 76 respondents were moderately to enthusiastically interested in participating in an introductory Python training course, especially College of Charleston students in the marine biology program and the academic/state researchers (Figure 1). The questionnaire also highlighted that most interest was at the introductory level (Figure 2).

[Insert Figure 1 here]

Figure 1. Interest in Python programming course development based on local affiliation. "CofC" is the College of Charleston, "GPMB Master's Student" are the Graduate Program in Marine Biology graduate students, "SCDNR" is the South Carolina Department of Natural Resources, "NOAA" is the National Oceanic and Atmospheric Administration and contains mostly responses from staff from the National Ocean Service's National Centers for Coastal Ocean Science Hollings Marine Science Laboratory, "NIST" is the local National Institutes of Standards and Technology staff, and "MUSC" are replies from Medical University of South Carolina personnel working on the Fort Johnson campus.

[Insert Figure 2 here]

Figure 2. Interest in Python programming course development based on programming experience.

Based upon the questionnaire responses, it was clearly recognized that respondents perceived a potential application of Python in their marine science work. This perception is further supported by evidence that some marine education programs are already training marine scientists in Python programming (for example, see https://datalab.marine.rutgers.edu/2020/11/introduction-to-python-data-analysis/ and http://mckays630.github.io/2015-05-04-usfcms/) for its applications in ocean science. On the Fort Johnson campus, the Python programming language has been used to develop a library Web application to visually promote new book purchases, assist in library collection analysis, quantify coral bleaching from photos, count the number of bacterial colonies on plates, process metagenomic DNA, automate web searches, and develop a prototype application to locate dolphins. There is anticipation that the need for Python programming will continue to grow in the upcoming years to support fishery management, data analysis, and

machine learning projects. It is thought that all these efforts could be supported with the development of local programming expertise providing a community of interested biologists and scientists using Python to support their analyses.

While there is great Python documentation available on the distribution website, in numerous outstanding books, through websites like Stack Overflow (https://stackoverflow.com), and in many online videos, the best option to meet the local need for programming expertise was determined to be localized training since it can sometimes offer the best results for developing local knowledge and can offer highly targeted coding examples pertinent to the interests of the students. In addition, offering a series of local courses would provide opportunities for students, faculty, and state/federal researchers to develop the professional coding networks which could help support a local Python community after the course. The workshop series was developed by Geoff Timms, Librarian for Marine Resources, College of Charleston Libraries, and supported by Jeff Guyon, Branch Chief, Key Species and Bioinformatics Branch, NOAA/NOS/NCCOS during the fall semester, 2021.

Course Design and Content

Learning Cohorts

To encourage a participatory and engaging learning experience, the training sessions were designed to accommodate a maximum of nine in-person participants, with two cohorts initially anticipated. Small cohorts were chosen to maintain a sense of informality about the training and to encourage learning through a supportive social presence and a cognitive presence enriched with participatory activities, which are two of the three key tenets of the Community of Inquiry model of education (Alman et al., 2012; Rausch & Crawford, 2012). Furthermore, the low instructor-

to-participant ratio would increase opportunities for individuals to engage with both the instructor and each other. Remote participation was not initially offered to participants to dissuade their adoption of a spectator role that is easily accomplished within the relative anonymity of webinars. An intimate and informal learning environment was provided to encourage participants to interact and learn from each other as well as from the instructors.

Web-based Python Platform

As a practical programming course, participants needed access to a Python environment for a positive active learning experience. Python is usually installed on a server or end-user computer, along with additional libraries and dependencies needed to achieve the user's programming goals. In addition, programmers often install Integrated Development Environment (IDE) software to efficiently write, test, and troubleshoot code. Workshop participants from Federal and State agencies frequently do not have the freedom to install software at will on their work computers. For this reason, a hosted Python environment, accessible on a web browser, was chosen to facilitate hands-on learning activities. A further advantage of a hosted environment is that once it is configured, all users work upon a consistent, common platform.

This introductory course assumed no prior programming experience, and it was thought that participants would be best served by supporting them to focus primarily on writing Python code rather than additionally trying to learn to use an IDE. Thus, Jupyter Notebooks were chosen as the environment in which participants would experience Python. Jupyter Notebooks, used within JupyterLab (https://jupyter.org/), are arranged such that a single column of cells can be designated as executable code, markdown (formatted text), or raw (unformatted) text. Each code cell may be executed independently of other cells, or all code cells may be executed in automatic

succession. Much like in an IDE, Python error messages are displayed, appearing immediately after the code cell in which an error occurs. The text cells can be used to provide supplemental information and explanations, as well as questions, tasks, or links to external resources. In addition to textual narrative, media like YouTube videos and images can be embedded in cells, while code can generate interactive visualizations within the notebook (Davies et al., 2020). While interactive notebooks function as a place to write code and analyze data, they also provide researchers the opportunity to document their research strategies and outcomes *in situ* (Perkel, 2018). These features make Jupyter Notebooks an ideal environment for participants to use during hands-on training sessions as well as a reference for independent use both before and after the scheduled meetings.

To serve executable Jupyter Notebooks to users in their browsers, the notebooks must be hosted on a server on which Python is installed and configured. For this course, the Jupyter Notebook files were maintained in a GitHub code repository (https://github.com/). Binder (https://mybinder.org/) was used to create a Docker image of the files in the GitHub repository, including reading a Python dependencies configuration file in which the version of Python and additional required libraries were specified. A JupyterHub server then served the Docker image as an interactive session in the user's browser, allowing the user to execute Python code. All these services were freely available and easy to configure; a distinct advantage over the potential cost and time commitment to manage one's own cloud based JupyterHub server (Kim & Henke, 2021). The Turing Way Community, in its handbook, provides concise guidance to quickly develop and launch this environment (The Turing Way Community et al., 2019).

Course Structure and Python Competencies

The course was designed to introduce participants to writing Python code and assumed no previous coding experience in any programming language. In addition to building a practical foundation of Python knowledge and experience, another desired outcome was to share and discuss potential applications of Python with specific examples pertinent to their research. Similar goals have been addressed in different ways in other programming courses. In one example, a course follows a logical progression from introducing Python syntax and basic programming competencies to applying them in common bioinformatics problems (Mariano et al., 2019). In an intensive four-day Python workshop, participants learned and exercised Python skills in morning sessions before attending afternoon seminars where different applications of Python were demonstrated by biologists from various subdisciplines. These were followed by group coding activities (Zuvanov et al., 2021). Making a more defined distinction between writing code and solving problems with code, a Computational Approaches for Life Scientists credit course in Israel had a programming prerequisite and introduced the Python language in the first six hours. The course subsequently focused on training students in structured computational thinking, where they applied Python to solve biological problems (Rubinstein & Chor, 2014). The workshop series described here focused significantly upon developing basic Python programming competencies, incorporating limited exposure to potential applications throughout.

Recognizing that many participants support fieldwork activities in early spring, a seven-week series of training sessions was designed, each of 90 minutes duration, to begin in the second week of January 2022. The session titles are listed below and the structure for each session, including the session learning outcomes, is outlined in Appendix 1: Session Descriptions.

Session 1 – Introduction to Jupyter Notebooks, data types and data structures

Session 2 – Getting content from iterables like lists and dictionaries

Session 3 – Creating and calling user-defined functions

Session 4 – Working with CSV files

Session 5 – Searching for string matches in data structures and matching patterns in data with regular expressions

Session 6 – Getting web-based data from Application Programming Interfaces (APIs)

Session 7 – Working with classes

Session Design

Bringing individuals together for training inevitably results in a group with varied learning experiences, preferences, and expectations. This is particularly true when participants represent multiple age groups spanning several decades and broad levels of academic attainment ranging from undergraduate degrees to post-doctoral qualifications. To stimulate participants' interest, each session utilized several teaching styles, engaging learners with observation, reflection/discussion, experimentation, and problem-solving activities. At the end of each session, participants were presented with a short multiple-choice quiz to complete before the next session. Quizzes were developed on the Springshare LibWizard quiz platform (https://springshare.com/libwizard/) and used to identify areas of participant confusion as well as to aid with the transfer of knowledge to longer-term memory, through recall.

Session content and the corresponding notebooks were organized to scaffold learning. In coding, active learning and critical thinking are two key processes to develop problem solving skills. Coding concepts were introduced through demonstrations and executable code,

participants were asked “what would happen if…” questions and the instructors fielded participant questions, and then the class transitioned into applying concepts with problem-based coding activities, including subsequent discussion. This progression places increasing responsibility upon the student to take ownership of learning to solve problems while scaffolding provides an appropriate support structure, with the goal of increasing independence in learning (Greening, 1998). Scaffolding also helps learners achieve success beyond their own existing abilities, which is particularly important in subjects where they have no previous experience (Grévisse et al., 2019). Participants in the course were able to access the Jupyter Notebooks in their own time after the class to help facilitate the learning process both through review and additional experimentation.

Inter-session scaffolding employed competencies acquired in previous sessions to reinforce accumulated knowledge through application as well as to aid retention. While the first two sessions introduced simple but essential concepts, they were reinforced in subsequent sessions in the context of other code. The fundamentals of iteration and conditional logic were introduced in Session 2 and were featured in almost every subsequent session. User defined functions, first introduced in Session 3, were also used in Sessions 5 and 7 to demonstrate their role in creating efficient and well-structured code. Regular reuse of fundamental coding concepts in problem-solving activities across multiple sessions enabled users to both recall and apply discreet competencies in functional code.

Even though this course was introductory, participants were trained to address and handle errors, equipping them with a key competency required to become independent in learning to write and test code. Programming languages are not static; Python, among other languages, is in continual development and syntax can change. Thus, participants learned to read and interpret

syntax error messages and, in some sessions, were also intentionally led into situations that resulted in value or logical errors. By learning to embrace error messages as a source of information about the nature and location of the error, participants gained insight into resolving coding problems.

Monitoring Participant Progress in Coding Activities

While hands-on coding activities were anticipated to be important active learning opportunities, they also presented challenges for managing the flow of each session. Individuals of diverse backgrounds, experiences, and preferred learning styles complete problem-solving tasks at different rates. From an instructor's perspective it is difficult to judge how each participant is progressing, especially when those who are struggling may be reluctant to reveal publicly that this is the case. To keep the session moving on schedule while being attentive to participants' needs requires instructor awareness of the progress of the class with the assigned activities.

Having previously used web-based virtual whiteboards in other contexts, a Miro board (https://miro.com) was created on an educator account to enable users to anonymously indicate completion of each activity. An instructor can create a Miro board with a fixed formatted background and instructions, while providing participants the ability to overlay virtual sticky notes, shapes, and text. Users can access the board in a web browser and use it without logging in, thus preserving anonymity. Laid out in a grid, the board for this course represented each session as a row and each coding activity as a labeled box within the row. Above the boxes were a series of colored "dots" where each color was affiliated with a learning cohort (see Figure 3). As users completed each activity, they were instructed to drag a dot of the appropriate color into the box for the activity. With this reference, the instructor can quickly assess how many people

in that cohort have completed a task, indicating if there is general understanding or confusion about the activity.

[Insert Figure 3 here]

Figure 3. Screenshot of a Miro board showing participant indicators (dots) of activity completion.

Supplemental Resources in an Online Guide

To support self-driven learning, a web-based guide was created on the college's LibGuides platform, where online library guides are hosted. The main guide page contains links to the GitHub repository and to launch the Jupyter Notebooks, as well as links to two Miro boards. A Question-and-Answer page was created to capture participant questions over the duration of the course and to provide answers for reference. Links to online resources like official Python documentation (https://docs.python.org/3/), Stack Overflow (https://stackoverflow.com/), glossaries, tutorials, biology-related APIs, etc. are found on a Resources page. One of each week's sessions was recorded using Zoom and uploaded to YouTube. A Recordings page on the guide contains each of these YouTube videos as embedded video players. The online guide served both as a launching point at the beginning of each training session as well as an ongoing reference for participants, including after the course finished. Access to the guide was password protected to prevent non-participants from accessing the quizzes and editing the Miro boards.

Running the Course

Registration

Two cohorts were initially scheduled on Tuesday mornings and Friday afternoons. Following promotion of the upcoming sessions in the fall of 2021, registration was launched using Eventbrite (https://www.eventbrite.com/) on November 29, 2021, limiting registration to nine participants per cohort. The 18 available seats were all booked within three hours. After receiving approximately 15 more inquiries about registration, a third cohort was scheduled on Wednesday mornings and offered registration directly to participants in chronological order of their registration inquiry. The third cohort was filled within 5 days.

The demographics of registrants are shown in Figures 4 and 5. Almost 60% were SCDNR researchers, with smaller representation from NOAA and marine biology graduate students. The most common level of academic attainment was 78% either holding (n=14) or currently working on (n=7) a master's degree and 11% holding a bachelor's or Ph.D. degree.

[Insert Figure 4 here]

Figure 4. Affiliations of course registrants.

[Insert Figure 5 here]

Figure 5. Educational attainment of course registrants.

Expressed Expectations of Participants

At the beginning of Session 1, participants were asked to answer the questions "What do you hope to achieve in this course?" and "How do you hope to use Python in the future?" using virtual sticky notes on a Miro board. Answers were coded and are found in Tables 1 and 2 respectively.

[Insert Table 1 here]

[Insert Table 2 here]

Adaptation of Format and Cohorts

Soon after registration of 27 participants was complete, some NOAA personnel indicated that they would still be working remotely in early 2022 and wanted to participate virtually. While other partner institutions' COVID-19 restrictions were alleviated enough to allow in-person participation, some personnel continued to enter precautionary quarantine at home due to potential exposure to the virus, so we agreed to offer remote participation. The college's Zoom platform (https://zoom.us/) was determined to be the web conferencing utility accepted for use by all partner institutions. To maintain the small cohorts, however, registration in each cohort was not increased beyond nine members.

As soon as the course started, other scheduling complications began to emerge that necessitated a flexible attitude towards the planned cohorts. Participants from SCDNR experienced the greatest schedule conflicts. Some participants were scheduled to attend a national fisheries conference being held locally. Then a mandatory safety training day was announced, preventing most members of a single cohort from attending Python training in their scheduled session that week. Several people became unwell and had to either miss a session or attend remotely. Lastly, mandatory fieldwork schedules were determined by supervisors, sometimes at short notice, and interfered with participation particularly during the latter part of the course.

In response to these challenges and to keep learning opportunities as open as possible, participants were informed that if they could not attend their cohort's session in any week, they were welcome to join any of the cohorts remotely. In addition, one session from each week was

recorded using Zoom and made available in the online guide. As a result, more people elected to participate remotely, and no session was attended in person by more than six people. Some participants used this freedom to choose each week which cohort's session best suited their schedule, reducing the constancy of the cohorts. Two or three participants stopped attending synchronous sessions and used the recordings as their point of participation. Ultimately, it was deemed more important that participants could attend a session each week than it was for them to participate in the same cohort each week.

Teaching a hybrid Zoom session where participants are participating both in-person and remotely impacts how an instructor conveys information and manages interaction. Furthermore, the instructor's attention becomes spread in multiple directions, including in-person engagement, virtual engagement by voice and by text, and ensuring that all participants are seeing the same material. This split attention can distract the instructor from teaching, coaching, and troubleshooting effectively. To alleviate this, Timms led sessions 1-6 in person and Guyon attended remotely (sessions 1-4) whereby he coordinated communications with Zoom participants and helped troubleshoot any technical difficulties. In the final session, Guyon instructed remotely and Timms provided in-person support in the classroom.

Observations with Problem-based Learning Activities

Short hands-on coding activities were designed to enable participants to apply what had just been learned, in a problem-based setting. Participants, however, developed conceptual understanding at different rates during sessions, resulting in varied abilities to design a coded response to the task. In some cases, individuals could not translate the task into computational thinking, whereby writing code would logically achieve the desired outcome. Usually, the task was very similar to

something previously demonstrated and discussed. In response, these activities were introduced with intentional reference to similarities with what had previously been learned. Furthermore, to guide participants towards a solution, the overarching framework of the solution was demonstrated without immediately completing all the code. For example, if the solution necessitated iterating over a dictionary to extract data, the instructor presented the initial "for" loop to trigger an association with previous learning. As time spent on the activity progressed, parts of the solution were added for participants to reference, either for comparison with their own code or for guidance about what to do next.

As participants tried to write their own code, they inevitably experienced syntax, variable referencing, and logical problems. For example, when writing loops or functions, participants regularly missed the colon at the end of the opening statement or did not correctly or consistently indent subordinate lines of code. The single "=" (variable assignment operator) and double "==" (equality evaluation operator) were also frequently used incorrectly. Reminders were provided about common mistakes both while introducing the activities and as the activities progressed, which helped participants preempt or resolve problems quickly. Logical and variable referencing problems presented a greater challenge. These required that the instructor look at the individual's code, which took focus away from the class as a whole and, in some cases, occupied excessive time to identify a problem. Having one instructor online helping remote participants was especially helpful, and small sections of code could be shared for troubleshooting in the Zoom chat.

Outcomes

Of 27 registered participants, one did not attend any sessions and one attended only the first session. The remaining 25 continued participating through the end of the course. The week after the course ended, an online survey was distributed to the 25 active participants. After two weeks, 15 responses were received and analyzed. The first question asked participants to indicate their preferred method for attending the training (see Figure 6). All but one response indicated a preference for live participation, although they were evenly distributed between attending in person and remotely. One participant indicated a preference for watching recordings.

[Insert Figure 6 here]

Figure 6. Answers from evaluations in response to the question, "What was your preferred method for participating in the Python training?"

Most of the remaining questions asked participants to use a Likert-type scale to respond to questions about each participation option, the content and design of the course, their confidence in continuing to learn Python, and their awareness of its potential application in their work (See Figure 7).

[Insert Figure 7 here]

Figure 7. Box and whisker plots showing ratings (1 = "Not at all" and 5 = "Very") selected in response to a series of Likert-scale questions.

A final Likert-type question asked respondents to indicate their level of interest in attending higher level Python courses focused on specific competencies. Four subject areas were listed, and respondents were invited to suggest other topics. The indicators of interest are presented in Figure 8.

[Insert Figure 8 here]

Figure 8. Box and whisker plots showing expressed level of interest (1 = "Not at all" and 5 = "Very") in attending future Python training focused on specific competencies.

Data from the submitted post-session quizzes are found in Figure 9. Completion of quizzes was highest in the first week at 80.7%, but the number of submissions diminished nearly every week, until only five submissions (20% participation) were received after the final session. The lowest score each week demonstrated an upward trend, while the standard deviation trended downwards.

[Insert Figure 9 here]

Figure 9. Box and whisker plots of quiz scores for each of seven training sessions.

Discussion

Based upon the positive feedback from 15 respondents, a 60% response rate, this course can be considered to have successfully provided a practical introduction to learning Python. In addition to the formal evaluation, much positive verbal feedback was expressed by several participants. It

must be acknowledged, however, that it is possible that any participants who may have had a less favorable perspective of the experience could have self-excluded their feedback by not completing the survey, even though it was anonymous.

The highest rated response, with a mean of 5, and the only statement that was rated unanimously, was about the responsiveness of the instructors. This is probably a function of the small class size as well as having both a primary instructor and a support instructor to respond to participants in most of the sessions. This is a feature of the course design that will be maintained in future offerings. Both the quality of preparation and the importance of hands-on coding activities were highly rated at means of 4.7. The latter is no surprise; participants in a programming course that involves hands-on activities benefit from pedagogical practice that is far superior to lecturing alone.

The lowest rated response was about the value of the post-session quizzes, with a mean of 3.7. The quizzes were included as a form of summative assessment, but some participants indicated verbally in classes that the multiple-choice questions were too hard. One participant indicated this in an open text response in the survey, but also qualified that, while difficult, they were valuable for recalling and retaining information. A diminishing number of quizzes were completed each week. Two participants indicated that time constraints outside of class caused them to not take the quizzes. Upon completion of each quiz, participants were presented with their score and identification of which questions were answered correctly or incorrectly. With hindsight, the quizzes would have been more useful to participants if incorrect answers were accompanied by automated hints or explanations to help them discern the correct answer. This feature is available in LibWizard quizzes.

Jupyter Notebooks were rated with a mean of 4.1 for being intuitive or easy to use. This rating can be interpreted by some textual comments, where frustration with the kernel being dropped necessitating re-running of code cells in the notebook was mentioned by several respondents. Similarly, the mean rating of 4.3 for the usefulness of the learning content should be interpreted in the context of participant expectations expressed at the beginning of the course. Those who aspired to learn Python for Geographic Information Systems or image analysis during this workshop series, for example, might have found the content too rudimentary.

Some of the challenges that were experienced could perhaps be mitigated with a different pedagogical approach. The varied rates at which participants understood programming concepts and could apply them in exercises made it difficult to set a pace for the sessions that would suit all learners. Flipped learning inverts the classroom lecture and homework model whereby students are introduced to new material in their own time and at their own pace, while engaging in hands-on activities and assignments in the classroom with the support of the teacher. While the learning material might be in the form of e-textbooks, documents, or videos, studies demonstrate student preference for online videos prepared by the teacher (Bakheet & Gravell, 2021).

The flipped model has been shown to improve student satisfaction and performance, although it, too, presents challenges when students fail to prepare before class or need assistance in viewing learning content (Akçayır & Akçayır, 2018). In studies of flipped learning in computer programming classes, student performance in the flipped learning environment increased compared both to control groups and previous iterations of a class taught in a traditional format (Ruiz de Miras et al., 2022; Taşpolat et al., 2021). While student feedback is generally positive about this style of learning in programming classes, it has also been suggested that the self-study

component can be less effective for more advanced programming topics, necessitating a blend of the traditional and the flipped classroom (Davenport, 2018).

The main barrier to implementing the flipped learning model at Fort Johnson is the time participants would need to invest outside of the classroom to learn programming concepts. Having observed participants achieve understanding at different rates, some people would clearly need to spend considerably more time than others to comprehend programming concepts. Most participants were practicing professionals with diverse roles and varying levels of autonomy in choosing their work activities and schedules. Graduate students must also negotiate existing course/lab work, employment, and research responsibilities to be able to participate. The time required to prepare for and attend each session could be prohibitive for some people to participate at all. This can also be true for a traditional course approach with homework assignments, as was experienced by graduate students in both a graduate Python course and an optional workshop series (Chapman & Irwin, 2015; Zuvanov et al., 2021).

The adaptation to a hybrid course was made in response to emerging participant needs. In the post-course evaluation, preferences for in-person and live online attendance options were equal at seven responses each. Hybrid training, with both in-person and remote participation, presents challenges for incorporating group work in the form of creative design and/or problem-solving activities. Peer collaboration is a valuable active learning method that has been used effectively in other programming courses, ranging from pair programming to small group work, and incorporating peer review of coding efforts (Chapman & Irwin, 2015; David, 2021; Porter et al., 2013). Furthermore, the use of pair programming in introductory undergraduate computer science courses has been demonstrated to contribute positively to student performance and retention (McDowell et al., 2006; Porter & Simon, 2013). More recently, distributed pair programming, using

tools developed for facilitating remote collaborative coding, has been evaluated positively by both high school and undergraduate participants in remote computer science courses (Bigman et al., 2021; Satratzemi et al., 2018). The software infrastructure for remote participant collaboration is evolving and could potentially be utilized in future programming courses offered at Fort Johnson.

One comment about the course noted that the installation and configuration of Python and an Integrated Development Environment (IDE) as well as using a command line terminal were not addressed. This was intentional both due to the complexity of navigating institutional policies for installing software on federal and state laptops, as well as the time needed to help all participants install software on disparate systems. However, to facilitate and encourage the ongoing programming and learning of course participants, at least an introduction to these concepts would have been useful. Python distributions such as Anaconda feature a Graphical User Interface (GUI), an IDE (Spyder, in the case of Anaconda), and convenient management of Python libraries and dependencies (David, 2021). This is something to consider including in a future version of the course.

Interest in the four suggested future sessions was varied. Respondents demonstrated greatest interest in data visualization and data manipulation/analysis. These subjects are broadly applicable in many areas of research, so it is logical that they were popular options. The greatest distribution of interest, including the lowest Likert score, related to using Python with RaspberryPi minicomputers. This is likely due to varied awareness of both what a RaspberryPi is and how it could be used in scientific research. Anecdotally, we were aware of two participants with specific interest in this. Moderate interest was expressed in image analysis and associated machine learning. No additional subjects were suggested by respondents.

In this Python training series, the trainers both had existing Python skills and were active programmers. But in libraries where no personnel possess sufficient programming knowledge or experience to offer a course like this, an opportunity for partnership exists to use existing knowledge in the broader academic or research community. For example, at academic institutions with computer science programs, faculty or graduate students may be available to provide or support a short programming course. The library might coordinate, sponsor, or host training sessions where the coding expertise is provided from elsewhere. Instructional librarians can often provide pedagogical insight to partner with a programmer to design an effective training program, including using available technology to maximum effect.

Recommendations

Feedback from participants confirms that the inclusion of information about setting up a Python environment is key for ongoing skill development and application of Python. Whether by including a training session on setting up a Python environment or by providing web-based tutorials, future iterations of this class should include guidance on installing Python, Python libraries, and an IDE. The Anaconda package is ideal for installing these components.

Course objectives and an outline of the curriculum should ideally be made available to participants before they register for the course. The provision of this information will help align participant expectations with what is being offered. In addition, a description of how teaching and learning will occur will ensure that participants enter the course aware of what to expect. This practice is standard in academic contexts and will potentially mitigate surprises and consequential reduced participation after the course begins (Zuvanov et al., 2021).

While it is difficult to make course content contextually relevant to all individuals due to the diverse research undertaken across all participants, there is potential to facilitate individuals using a sample of their own data in the course during some active learning exercises. In addition, this would be particularly appropriate in higher level courses on data processing with the Pandas library and data visualization with the Matplotlib library. Use of real datasets should be incorporated into the session design wherever possible. Participants want to understand how the training can positively impact their work, so the connection to real data is important.

Maintaining momentum is important in learning and maintaining knowledge about programming. In response to participant feedback, expressed interest in future learning, and instructor observations, the following recommendations are considered a priority for the ongoing development of Python training at Fort Johnson:

- Offer a session on installing a Python environment to previous introductory course participants
- Offer a revised introductory course in Winter 2022, when the fieldwork season is over.
    - Publish course outline with registration so that participants can align expectations with what is covered
    - Try to group participants by programming experience level
    - Ask participants to bring sample data to use, or design a case study with contextually relevant data
- Develop and offer two short courses and resource guides for data visualization (Matplotlib) and data processing (Pandas)

- Build instructor experience with image and video analysis and machine learning, to showcase their application in marine science and generate interest in learning these competencies.
- Follow up with participants after the course to keep momentum going
- Develop a local Python users' group to encourage communication and collaboration in programming activities.

Last, due to the occasional problems experienced with the hosted JupyterHub including slow launch speeds and service downtime, exploration of self-hosting a Jupyter Notebook server or using a different service is recommended.

Conclusion

As programming competencies are increasingly required for scientists to process and analyze the volumes of data generated by research, Python has emerged as a popular general purpose programming language that can support a variety of data processing operations. The syntax of Python reduces barriers to learning this language and the many freely available web-based tutorials and discussion boards facilitate independent learning. While a competent Python programmer must ultimately be able to consult the official Python documentation as well as these other resources, the purpose of introductory courses such as the one described in this paper is threefold. Course participants must become aware of the capabilities and potential applications of Python, learn and apply the fundamental competencies of programming with Python, and become aware of resources to help them become independent in expanding their own skillset. To

encourage budding programmers to maintain momentum ongoing communication and facilitation of networking with peer programmers is recommended as part of a training program.

U.S. National Oceanic and Atmospheric Administration Disclaimer: The scientific results and conclusions, as well as any opinions expressed herein, are those of the author(s) and do not necessarily reflect the views of NOAA or the Department of Commerce. The mention of any commercial product is not meant as an endorsement by the Agency or Department.

Works Cited

Akçayır, G., & Akçayır, M. (2018). The flipped classroom: A review of its advantages and challenges. *Computers & Education*, *126*, 334–345. https://doi.org/10.1016/J.COMPEDU.2018.07.021

Alman, S. W., Frey, B. A., & Tomer, C. (2012). Social and cognitive presence as factors in learning and student retention: An investigation of the cohort model in an iSchool setting. *Journal of Education for Library and Information Science*, *53*(4), 290–302.

Badenhorst, M., Barry, C. J., Swanepoel, C. J., van Staden, C. T., Wissing, J., & Rohwer, J. M. (2019). Workflow for data analysis in experimental and computational systems biology: Using Python as 'glue.' *Processes, 7*(7), 460. https://doi.org/10.3390/PR7070460

Bakheet, E. M., & Gravell, A. M. (2021). Would flipped classroom be my approach in teaching computing courses: Literature review. *2021 9th international conference on information and education technology, ICIET 2021*, 166–170. https://doi.org/10.1109/ICIET51873.2021.9419631

Bigman, M., Roy, E., Garcia, J., Suzara, M., Wang, K., & Piech, C. (2021). PearProgram: A more fruitful approach to pair programming. *Proceedings of the 52nd ACM technical symposium on computer science education*. https://doi.org/10.1145/3408877

Casanova-Arenillas, S., Rodríguez-Tovar, F. J., & Martínez-Ruiz, F. (2020). Applied ichnology in sedimentary geology: Python scripts as a method to automatize ichnofabric analysis in marine core images. *Computers & Geosciences*, *136*, 104407. https://doi.org/10.1016/J.CAGEO.2020.104407

Chapman, B. E., & Irwin, J. (2015). Python as a first programming language for biomedical scientists. *Proceedings of the 14th Python in science conference*. https://doi.org/10.25080/MAJORA-7B98E3ED-002

Davenport, C. (2018). Evolution in student perceptions of a flipped classroom in a computer programming course. *Journal of College Science Teaching*, *047*(04). https://doi.org/10.2505/4/jcst18_047_04_30

David, A. A. (2021). Introducing Python programming into undergraduate biology. *The American Biology Teacher*, *83*(1), 33–41. https://doi.org/10.1525/ABT.2021.83.1.33

Davies, A., Hooley, F., Causey-Freeman, P., Eleftheriou, I., & Moulton, G. (2020). Using interactive digital notebooks for bioscience and informatics education. *PLOS Computational Biology*, *16*(11), e1008326. https://doi.org/10.1371/JOURNAL.PCBI.1008326

Ekmekci, B., McAnany, C. E., & Mura, C. (2016). An introduction to programming for bioscientists: A Python-based primer. *PLOS Computational Biology*, *12*(6), e1004867. https://doi.org/10.1371/JOURNAL.PCBI.1004867

Greening, T. (1998). Scaffolding for success in problem-based learning. *Medical Education Online*, *3*(1), 4297. https://doi.org/10.3402/meo.v3i.4297

Grévisse, C., Rothkugel, S., & Reuter, R. A. P. (2019). Scaffolding support through integration of learning material. *Smart Learning Environments*, *6*(1), 1–24. https://doi.org/10.1186/S40561-019-0107-0

Kim, B., & Henke, G. (2021). Easy-to-use cloud computing for teaching data science. *Journal of Statistics and Data Science Education*, *29*(sup1), S103–S111. https://doi.org/10.1080/10691898.2020.1860726

Lewis, K. P., vander Wal, E., & Fifield, D. A. (2018). Wildlife biology, big data, and reproducible research. *Wildlife Society Bulletin*, *42*(1), 172–179. https://doi.org/10.1002/wsb.847

Mariano, D., Martins, P., Helene Santos, L., & de Melo-Minardi, R. C. (2019). Introducing programming skills for life science students. *Biochemistry and Molecular Biology Education*, *47*(3), 288–295. https://doi.org/10.1002/BMB.21230

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, *49*(8), 90–95. https://doi.org/10.1145/1145287.1145293

Perkel, J. M. (2018). Why Jupyter is data scientists' computational notebook of choice. *Nature*, *563*(7729), 145–146. https://doi.org/10.1038/D41586-018-07196-1

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013). Success in introductory programming. *Communications of the ACM*, *56*(8), 34–36. https://doi.org/10.1145/2492007.2492020

Porter, L., & Simon, B. (2013). Retaining nearly one-third more majors with a trio of instructional best practices in CS1. *SIGCSE 2013 - Proceedings of the 44th ACM technical symposium on computer science education*, 165–170. https://doi.org/10.1145/2445196.2445248

Rausch, D. W., & Crawford, E. K. (2012). Cohorts, communities of inquiry, and course delivery methods: UTC best practices in learning-the hybrid learning community model. *Journal of Continuing Higher Education*, *60*(3), 175–180. https://doi.org/10.1080/07377363.2013.722428

Rubinstein, A., & Chor, B. (2014). Computational thinking in life science education. *PLOS Computational Biology*, *10*(11), e1003897. https://doi.org/10.1371/JOURNAL.PCBI.1003897

Ruiz de Miras, J., Balsas-Almagro, J. R., & García-Fernández, Á. L. (2022). Using flipped classroom and peer instruction methodologies to improve introductory computer programming courses. *Computer Applications in Engineering Education*, *30*(1), 133–145. https://doi.org/10.1002/CAE.22447

Satratzemi, M., Xinogalos, S., Tsompanoudi, D., & Karamitopoulos, L. (2018). Examining student performance and attitudes on distributed pair programming. *Scientific Programming*, *2018*. https://doi.org/10.1155/2018/6523538

Saunders, J. K., Gaylord, D. A., Held, N. A., Symmonds, N., Dupont, C. L., Shepherd, A., Kinkade, D. B., & Saito, M. A. (2020). METATRYP v 2.0: Metaproteomic least common ancestor analysis for taxonomic inference using specialized sequence assemblies - Standalone software and web servers for marine microorganisms and coronaviruses. *Journal of Proteome Research*, *19*(11), 4718–4729. https://doi.org/10.1021/acs.jproteome.0c00385

Taşpolat, A., Özdamli, F., & Soykan, E. (2021). Programming language training with the flipped classroom model. *SAGE Open*, *11*(2), 215824402110214. https://doi.org/10.1177/21582440211021403

The Turing Way Community, Arnold, B., Bowler, L., Gibson, S., Herterich, P., Higman, R., Krystalli, A., Morley, A., O'Reilly, M., & Whitaker, K. (2019). Zero-to-binder. In *The Turing Way: A Handbook for Reproducible Data Science*. Zenodo. https://the-turing-way.netlify.app/communication/binder/zero-to-binder.html

Voskoglou, C., Witkowski, J., Stephens, J., Korakitis, K., & Muir, R. (2021). *Developer economics, State of the developer nation 20th Ed., Q1 2021*. https://developer-economics.cdn.prismic.io/developer-economics/dbf9f36f-a31a-440a-9c22-c599cc235fa4_20th+edition+-+State+of+the+developer+Nation.pdf

Zuvanov, L., Garcia, A. L. B., Correr, F. H., Bizarria, R., da Costa Filho, A. P., da Costa, A. H., Thomaz, A. T., Pinheiro, A. L. M., Riano-Pachón, D. M., Winck, F. V., Esteves, F. G., Margarido, G. R. A., Casagrande, G. M. S., Frajacomo, H. C., Martins, L., Cavalheiro, M. F., Grachet, N. G., da Silva, R. G. C., Cerri, R., … dos Santos, R. A. C. (2021). The experience of teaching introductory programming skills to bioscientists in Brazil. *PLOS Computational Biology*, *17*(11), e1009534. https://doi.org/10.1371/JOURNAL.PCBI.1009534

Appendix 1: Session Descriptions

Session 1 – Introduction to Jupyter Notebooks, data types and data structures

The first thirty minutes of Session 1 was set aside to introduce participants and instructors, review the goals of the course, obtain feedback about participants' expectations, and introduce an online guide from which each week's Jupyter Notebook could be opened. A brief introduction to the Jupyter Notebook environment was then reinforced by its use during the remainder of the session and all subsequent sessions.

Learning Outcomes

- Understand basic data types and structures
- Test for data types
- Create, modify, and read lists and dictionaries

Session Content

- Annotating code with comments
- Introduction to integers, floats, and strings
- Introduction to variables
- Printing variables with the print() function
- Testing variable data types with the type() function
- Introduction to lists
    - List indexes
    - Testing list lengths with the len() function
    - Slicing lists

    - Modifying items in lists
    - Modifying lists with the .append(), .insert(), and .pop() methods
- Introduction to Dictionaries
    - Obtaining a dictionary's keys and values with the .keys(), .values(), and .items() methods
    - Modifying items in dictionaries
    - Adding items to dictionaries
    - Removing items from dictionaries with the .pop() method

Session 2 - Getting content from iterables like lists and dictionaries

Learning Outcomes

- Understand when looping is needed
- Design a loop
- Use logic to achieve desired outcome in each iteration of a loop

Session Content

- Indentation in Python
- The For loop
    - Iterating over a list
        - Iterating to retrieve list values
        - Iterating to retrieve list values and indexes with the enumerate() function

        - Nested lists
        - Iterating over nested lists with nested loops
    - Iterating over a dictionary
    - Iterating to retrieve keys, values, and both keys and values
    - Iterating over mixed data structures (combination of nested lists and dictionaries)
- The While loop
    - Creating a counter and counter conditions
    - Comparison operators (<, <=, ==, >=, >, !=)
    - Conditional statements and logical flow with if…elif…else statements
    - Logical operators in conditional statements (introducing and & or)

Session 3 - Creating and calling user-defined functions

Learning Outcomes

- Understand why user defined functions are useful
- Design a function and its parameters
- Use a function with appropriate arguments

Session Content

- Introducing user defined functions and demonstrating their role in code
- Creating a function with a parameter
- Calling a function and passing an argument

- Returning data from a function
- Functions with multiple parameters
- Functions with default arguments
- Calling a function with keyword arguments
- Local and Global variables

Session 4 - Working with CSV files

Learning Outcomes

- Understand how to read and write CSV file content
- Design loops and logic to access CSV data
- Evaluate errors and resolve data type problems with CSV data
- Design data structures to write data to a CSV file

Session Content

- Importing non-core libraries
- Exploring the csv library with the dir() function
- Using the help() function to understand specific functions
- Reading a CSV file
    - Opening a CSV file for reading in the context of the 'with' statement
        - Parameters used with file opening
        - Creating a csv.reader object

            - Iterating over a csv.reader object
            - Converting a csv.reader object to a list with the list() function
        - Creating a csv.DictReader object
            - Converting a csv.DictReader object to a list with the list() function
            - Iterating over a list of dictionaries to retrieve data
- Characteristics of data from CSV files
    - Determining the type of an empty cell
    - Replacing/removing characters from strings with the .replace() method
    - Casting a string as an integer with the int(function)
    - Rounding floats resulting from mathematical operations with the round() function
- Writing a CSV file
    - Opening a CSV file for writing in the context of the 'with' statement
    - Creating a csv.writer object
        - Writing a single row using a list with the .writerow() method
        - Writing multiple rows using nested lists with the .writerows() method
    - Creating a csv.DictWriter object with a list of field names as column headings
        - Writing column headings with the .writeheader() method
        - Writing multiple rows from a dictionary with the .writerows method
        - Handling default values for missing data with DictWriter 'restval' parameter
        - Raising errors for unanticipated keys in the data with DictWriter 'extrasaction' parameter

Session 5 - Searching for string matches in data structures and matching patterns in data with regular expressions

Learning Outcomes

- Understand the concept of membership
- Design tests for membership in different data types/structures
- Understand pattern matching
- Design a regular expression
- Evaluate results of pattern matching functions

Session Content

- Membership and the Python ‘in’ operator
    - Using the in operator with conditional statements
    - Testing for membership in strings
    - Testing for membership in lists and in strings in lists
    - Testing for membership in dictionaries and in dictionary keys and values
- Regular expressions
    - Importing the re library
    - Simple matching of a specific string
        - Searching for a (the first) match with the re.search() function
            - The match object and its attributes
            - Retrieving the match with the .group() method

            - The NoneType and testing for NoneType
        - Searching for all matches with the re.findall() function
            - The re.findall() response (a list) with and without matches
            - Counting matches with the len() function
        - Searching for all matches with the re.finditer() function
            - Iterating over the matches object
            - Retrieving the matching pattern with the .group() method
    - Constructing a regex to match multiple word endings
    - Constructing a regex to match multiple patterns
    - Using the Pythex (https://pythex.org/) tool to design and test regexes
    - The Pythex regex cheat sheet

Session 6 - Getting web-based data from Application Programming Interfaces (APIs)

Learning Outcomes

- Understand the purpose of an API
- Evaluate the structure of an API
- Design a requests query for an API
- Understand the format of API data
- Access and use API data

Session Content

- Importing the requests library
- The get and post methods of HTTP requests
- The World Register of Marine Species (WoRMS) API structure and documentation
    - Constructing requests to various applications of WoRMS
    - Formatted strings and incorporation of variables
    - Introduction to JSON
    - Response objects and the .url, .text, and .status_code attributes
    - Decoding JSON data with the .json() method
    - Determining the structure of returned data
    - Designing a recursive function to iterate over predictable data structures of unpredictable depth
- The NOAA Tides and Currents API structure and documentation
    - Passing key value pairs in a URL
    - Constructing a request including data in key value pairs, using a dictionary and the ‘params’ parameter
    - Visualizing complex output clearly with the pprint library.
    - Teaser about plotting data with the matplotlib.pyplot library
    - Requesting and handling responses in different formats (JSON, XML, and CSV)
- An overview of authentication and APIs

Session 7 - Working with Classes

Learning Outcomes

- Understand the concept of Object-Oriented Programming
- Understand instance variables and attributes
- Understand class and static methods

Session Content

- Making an object
- Creating a blank class instance and assigning variables
- Defining a simple class with variables and functions
- Class and Instance Variables
- Class variables - list concerns
- Fixing the class list variable issue by using an instance variable
- Class and instance variables - integers
- Use of class init to assign values
- Working with Getters and Setters - ways to control access to private variables and validate input
- Assigning new class operands
- Inheritance - how to include one class within another
- Use of class methods to make a class instance
- Use of static methods to make a namespace
- Use of dataclass