Abstract

The rise of “big data” within the biological sciences has resulted in an urgent demand for coding skills in the next generation of scientists. To address this issue, several institutions and departments across the country have incorporated coding into their curricula. I describe a coding module developed and deployed in an undergraduate parasitology course, with the overarching goal of familiarizing students with the Python programming language. The module, which was completed over four days, aimed to help students become comfortable with the command line; execute summary statistics and Student’s t-tests through coding; create simple bar and line graphs using code; and, parse, handle, and analyze imported data sets. There is currently no standard “best practice” for teaching coding skills to biology majors, but this module can serve as a template to ease students into coding, and can then be modified and built out for teaching more advanced skills.

Key Words: coding; statistics; data; science; parasitology.

Introduction

The biological sciences, like many other scientific fields, are currently undergoing a “big data” revolution (McCulloch, 2013). The pace at which ecological and molecular data are being collected, thanks to advances in technology, is providing exciting insights into natural phenomena, but it also poses serious challenges for data analyses (Li & Chen, 2014). As early as two decades ago, simple spreadsheet-style programs such as Excel were considered sufficient for analyzing data sets (Li & Chen, 2014). As early as two decades ago, simple spreadsheet-style programs such as Excel were considered sufficient for analyzing data sets. Today, whole genomes generated through next-generation sequencing and millions of data points collected via remote-sensing instruments mean that scientists must now incorporate advanced coding skills to automate traditional statistical analyses and even partition the processing of computationally demanding data sets (Barone et al., 2017). Collectively, these types of analyses are nested within the broad field known as “data science,” and it is imperative that the next generation of biologists are equipped with a suite of scripting skills. Improving quantitative skills is one of six core competencies outlined in the AAAS’s Vision and Change manifesto (Brewer & Smith, 2011), and the development of coding skills will aid in fulfilling this competency.

Because of the broad nature of the biological sciences, workshops and projects that seek to introduce coding to biology majors are usually discipline specific. However, one area that is standardized across all biological fields is statistics. Calculations of averages and standard deviations (along with execution of t-tests, ANOVAs, regressions, etc.) are completed in the same manner. Therefore, one method for introducing biology majors to coding can be through the medium of statistical analyses. Such skills can be transferred to other courses that emphasize data analyses, and they can be of practical use to students who are currently working in a research lab and are analyzing their own data sets. Indeed, incorporating such practical skills in the classroom can fully prepare an undergraduate for the research lab environment (David, 2018).

Currently, R is the most popular programming language used by biologists for data analyses (Gentleman, 2008). Because of its open-source nature and its community of more than 2 million users, R has been incorporated into a number of undergraduate ecology curricula (Auker & Barthelmess, 2019). Python, another programming language, is also open source and boasts an even larger community (Guo, 2014). In addition, scripting (i.e., writing the code) is arguably easier in Python than in R. Python’s simple commands (Ozgur et al., 2017) make it more efficient as a “learner programming language” than R (Ozgur et al., 2017). Finally, unlike R, Python is a “complete” language, which means that it is regularly used in product development and implemented in machine learning. Therefore, students who choose to further their exploration of Python will have a highly sought-after skill set for a wide variety of data science careers. Here, I present a multipart Python learning module that was incorporated into an undergraduate parasitology course.
Learning Goals & Objectives

The overarching aim of this module was to familiarize undergraduate biology majors with the Python programming language, using statistical analyses as an instructional medium. This project was divided into four sequential phases (Table 1).

Class Profile

This module was deployed in fall 2019 in an undergraduate parasitology course at Clarkson University. Class size was 16 students, all of whom were junior and senior biology majors who had previously taken at least one course in statistics. The classroom space included a large round table where students could directly interact with each other. The module requires a dedication of three class periods, 75 minutes each.

Python Installation

The students were required to use their own laptops for the duration of the module. For students who did not have access to laptops, loaner laptops were secured prior to course commencement. To create a more inclusive environment, it is strongly suggested that each student be contacted individually to determine any hardware needs for completing this module. To ease students into Python, the graphical user interface (GUI) known as Anaconda (similar to R-Studio) was used. Anaconda tracks the outputs of a code, and students are immediately prompted when an error is made while scripting.

Table 1. Learning objectives for Python programming exercise

| Instructional Phase                                | Learning Goals |
|---------------------------------------------------|----------------|
| Day 1: Familiarity with the Command Line           | LG 1: Type and execute a series of one-line codes on the Python GUI. |
| Day 2: Introduction to Python Libraries and Basic Statistical Analyses | LG 2.1: Organize a data set into a data frame using the pandas library.  
|                                                   | LG 2.2: Calculate summary statistics using the pandas library.  
|                                                   | LG 2.3: Execute a pairwise t-test using the scipy library. |
| Day 3: Simple Graphing in Python                  | LG 3: Create a line and bar graph using the matplotlib library. |
| Day 4: Importing and Analyzing a Raw Data Set     | LG 4.1: Import and parse an Excel data set using pandas.  
|                                                   | LG 4.2: Calculate summary statistics and depict the data on a bar graph using pyplot.  
|                                                   | LG 4.3: Execute a t-test using scipy. |

Phase 1: Familiarizing Students with the Command Line

The command line is the “bread and butter” of a programmer’s life. From the command line, some of the most complex analytic problems in the world can be solved. For a student with no prior experience in coding, becoming comfortable using this command line is the first crucial task, and like any language, it requires practice. As a “rite of passage” for the beginning programmer, the first line of code a student learns is one that commands the phrase “Hello World” to be printed on the screen. In the command line, students should be instructed to type the following command and then execute the code using their F9 hotkey:

\[
\text{print(‘Hello World’)}
\]

If the code has been written properly (apostrophes, brackets, no extra punctuation), the output on the console should read: Hello World. The ‘___’ characters define the variable by turning it into what is known as a string, which can then be read via the print function. Students should spend a minute or two practicing with the print command. Other one-line code exercises that were used involved adding strings together using the ‘+’ command, for example:

\[
\text{print(‘Biology Majors’ + ‘ Are Awesome’),}  
\]

OUT: Biology Majors Are Awesome

While students familiarize themselves with typing in the command line, they should also understand that errors occur frequently when coding. In fact, scripting errors are unfortunately the norm rather than the exception, even among the most seasoned programmers. The good thing about using the Anaconda platform to run Python is its ability to identify syntax errors in a block of code (i.e., a script) and then direct the coder to the problem via a warning prompt – a feature extremely helpful to beginners. To illustrate how a syntax error can arise, students
should try printing the previous phrase but omitting one of the apostrophes, so:

```
print('Hello World')
```

When the code is executed using the F9 hotkey, the output will be shown on the IPython console (Figure 2).

Here, students learn one of the most frustrating aspects of writing code, which is that errors can arise due to what we perceive as a minor typo (a missing apostrophe, a missing bracket, two spaces between two commands instead of one, etc.). However, what we perceive as a slight error is enough to render an entire block of code unreadable in the computing realm. To put this into context, imagine a programmer at a large software company writing an important commercial program or application composed of a block of code 800 lines long. The absence of a bracket on line 445 is usually sufficient to render the entire script unintelligible, which will likely cause the program to crash.

**Phase 2: Introduction to Python Libraries & Basic Statistical Analysis**

For data analytics, Python has a vast array of libraries (e.g., pandas, scipy, numpy) that can be imported to execute various statistical analyses. The results can then be visualized in an almost countless number of ways. Mastering the tools of just a single library can take years, and students should be encouraged to explore them further.
First, to organize the data into a table in Python, students import the pandas library. Have students write the code out line by line (Figure 3). Students should pay attention to the fact that two types of brackets are being used: [ ] and { }. Mixing these up or using one before the other is enough to crash the program. Students should also notice the pound symbol (#) that was added at the end of each line (arrow), followed by a message in gray font. Python does not read anything written after the # symbol, so it is often used to annotate scripts. By annotating a script, you explain to other coders exactly what you were trying to accomplish line by line, which allows for easier troubleshooting in a coding community. In fact, adding these explanations is common etiquette in coding and that they should try to do so for all their coding projects. All seven lines of the code should be executed separately using the F9 hotkey. The output should appear on the console (Figure 4).

Based on the outputs, students should now see that the mean pretest score was 5.00 ± SD 13.66 and the mean posttest score was 61.60 ± SD 24.91.

Finally, to determine whether there was a significant difference in the means, instructors should first review the theory behind Student’s t-test (ideally, students should already have completed at least one statistics course, so the review can be a brief refresher). To execute a Student’s t-test to compare the two means, students must first import another library, scipy, and then type in an additional three lines of code. In the Python command line, the script and output should appear as in Figure 5.

The output provides both the test statistic and the P-value. Based on this information (P < 0.05), we can reject the null hypothesis and conclude that the posttest scores were significantly higher than the pretest scores.

### Phase 3: Basic Graphing in Python

Now that students have become familiarized with the command line and are able to import libraries into the Anaconda platform, another important skill is the ability to build graphs by writing code. One of the biggest advantages of building graphs using programming languages such as R or Python, compared to those produced by Excel, SPSS, SAS, and others, is that you have complete control over the minutest detail of the graph. There is no “default” graph – you build it from the ground up. Here students will learn how to build (1) a simple line graph and (2) a simple bar graph. In the Python command line, the script should appear as shown in Figure 6.

Students will notice from looking at the script that creating bar charts using matplotlib is basically the same as with line graphs, except that one may choose to center the bars using the “align” command. The full block of code and the corresponding graph, when executed, will appear as in Figure 7.

Of course, the majority of bar graphs that many biology students will use involve clear independent variables along with the dependent variables. To convert integers on the x and y axes to categorical variables such as “low” or “high,” the code must be altered as shown in Figure 8. Students should reflect on the new lines of code that were added and think carefully about how they alter the output.
Phase 4: Importing Raw Data Sets & Synthesis

For biologists, the main appeal of learning how to code is the capability of handling large data sets, which can be partitioned and analyzed in various ways. In this final exercise, students will learn how to import data from a generic Excel spreadsheet, partition the data in several ways using the Python data frame function, analyze the data using statistical tests, and create a graph to illustrate the data. Because this module was deployed in a parasitology course, the synthesis assignment was based on material that was covered in lecture, specifically the effects of anti-helminth drugs on trematode parasites.

For this exercise, students work in pairs and divvy up the workload (which is essentially what programmers do before handling a large project) (Figure 9). Prior to class, students are given an Excel

![Image](image-url)
Figure 6. (Left) Screenshot showing script for plotting a line graph. (Right) Screenshot showing output of executing that script.

Figure 7. (Left) Screenshot showing script for plotting a bar graph. (Right) Screenshot showing output bar graph based on execution of that code.

Figure 8. (Left) Screenshot of script for producing a detailed bar graph. (Right) Screenshot of the bar graph produced from that code.
In 2005 a pharmaceutical company designed an improvement of the drug known as ‘Praziquantel’ to combat what appears to be a novel strain of *Schistosoma mansoni*, which recently emerged in a heavily infested province in Colombia. The drug is currently in the earliest stages of human experimental treatment. A recent four-year trial of the drug assessed its effect on prevalence in ten populations. The trial was repeated each year for four years in a village that had access to clean water and sanitary conditions. The raw data was assembled, compiled and made available to the academic community after the study was published.

A CSV file containing the raw data collected from the trial was uploaded to Moodle.

1. Import the excel data into a Python data frame
2. Working in pairs, discuss how you plan to partition the data in python and what type of analysis you will use.
3. Carry out the appropriate analysis and depict the results on a graph.
4. In a paragraph, discuss the results.

To complete the assignment:
Copy and paste your scripts into a text editor (e.g., notepad), save the file and submit to odowd@Clarkson.edu. Results explanation can be submitted in MS-Word.

spreadsheet with the data saved in a CSV format (csv instead of xls) (Figure 9). CSV stands for “comma-separated values” and differs from XLS files in that the data are arranged in plain text without unnecessary formatting and layout information. This allows for easier viewing, editing, and parsing of data. The extra “fluff” found in XLS files can corrupt the data set by filling it with unexpected characters.

To import the Excel file into Python, students must first move the file to a destination that can easily be retrieved (e.g., the desktop). The filename should not include any spaces or unnecessary characters that would conflict with other parts of the code. Students in the parasitology class agreed on “SchistosomaData” as the file name, and it was saved to the desktop. The import code and the corresponding output are shown in Figure 10.

Now that the file has been imported into Python, we can arrange the data in different ways (by treatment year, by treatment, or by population). At this point, students working in pairs should begin discussing their preferred way to arrange the data. In the parasitology class, the most popular choice was grouping the prevalence data by treatment regardless of year or population. To arrange the data by treatment, the code shown in Figure 11A was used.

By executing the line of code in Figure 11A, we have created two pools of data: “Placebo” and “Praziquantel.” To generate summary statistics for both treatments, a derivative of stat code used earlier – describe () – can be added in the line (Figure 11B).

Once this code is executed, the output on the console should appear as in Figure 11C.

From the output, the mean (± SD) parasite prevalence in both treatments has been calculated: mean prevalence in populations treated with the placebo was 0.65 ± 0.04, compared to 0.61 ± 0.07 for those treated with praziquantel.

Next, students can work on their own to create a simple bar graph to illustrate the results.
Almost all the groups in the parasitology class were able to devise a script on their own to build the graph. An example script and its corresponding output are shown in Figure 12.

At this point, students can carry out a t-test on the prevalence data to determine whether praziquantel had a significant effect in reducing parasite prevalence compared to the placebo after four years of continuous treatment. As they did for the bar graph, groups should discuss how best to rearrange the code they learned earlier in the module to apply to this problem. Some groups in the parasitology class queried online sources for help, and many of them discovered Stack Exchange (a self-moderating network for coding) on their own. Using this information, several groups were able to construct three-line and four-line coding solutions from the scipy library. These were shared with the rest of the class. For the sake of brevity, one solution is shown in Figure 13 (left panel). This code breaks down the prevalence data by treatment – similarly to the code used earlier to organize the data frame, but with the difference that this code also defines the two independent data sets as “Placebo” and “Praziquantel” using “==” (an equality operator that checks whether two values are equal). Because the variables have now been defined, they can be compared using the standard t-test function from the scipy module. If the code is successfully executed, the output should appear as in Figure 13 (right panel).

According to the result, the P-value is 0.033 (<0.05), which indicates that parasite prevalence was significantly lower in the drug-treated population than in the control. For the final part of the project, students worked together as a class in providing explanations for the results. Some of these involved discussions on whether administering the drug will make a difference despite the fact that prevalence was statistically lower than in the praziquantel-treated population. Others discussed rearranging the code to analyze the effect of the drug from year 1 to year 4 to investigate whether drug resistance is a potential issue.
Final Thoughts

In conclusion, while this module had clear deliverables and was somewhat structured, it was also open ended, in that students were allowed to use the command line as a sandbox for developing their coding skills. This was achieved by incentivizing the module and providing extra credit for successful completion. The module was not graded, so the focus was set purely on learning to code rather than reaching a distinct end point. Of course, the next step would be for students to apply their new coding skills in a research setting, similar to what was recently done by Rahn et al. (2019) in an undergraduate introductory biology course at William and Mary. While this project was implemented in a small course, it can also be scaled up for larger classes by incorporating teaching assistants, who can provide an added scaffold for the duration of the project.

References

Auker, L.A. & Barthelmess, E.L. (2019). Teaching R in the undergraduate ecology classroom: approaches, lessons learned, and recommendations. bioRxiv, 666768.

Barone, L., Williams, J. & Micklos, D. (2017). Unmet needs for analyzing biological big data: a survey of 704 NSF principal investigators. PLoS Computational Biology, 13, 10.

Brewer, C.A. & Smith, D. (2011). Vision and Change in Undergraduate Biology Education: A Call To Action. https://live-visionandchange.pantheonsite.io/wp-content/uploads/2011/03/Revised Vision and Change Final Report.pdf

David, A.A. (2018). Pedagogy helps to prepare undergraduates for the research lab. Nature, 561, 177.

Gentleman, R. (2008). R Programming for Bioinformatics. Boca Raton, FL: CRC Press.

Guo, P. (2014). Python is now the most popular introductory teaching language at Top US Universities. Communications of the ACM Blog (BLOG@CACM), July (2014).

Li, Y. & Chen, L. (2014). Big biological data: challenges and opportunities. Genetics, Proteomics & Bioinformatics, 12, 187.

McCulloch, E.S. (2013). Harnessing the power of big data in biological research. BioScience, 63, 715–716.

Ozgur, C., Colliau, T., Rogers, G., Hughes, Z. & Myer-Tyson, B. (2017). Matlab vs. Python vs. R. Journal of Data Science, 15, 355–372.

Rahn, J., Willner, D., Deverick, J., Kemper, P. & Saha, M. (2019). Incorporating computer programming and data science into a guided inquiry-based undergraduate ecology lab. American Biology Teacher, 81, 649–657.

Andrew A. David is an Assistant Professor of Biology and Director of First-Year Biology in the Department of Biology, Clarkson University, Potsdam, NY 13699; e-mail: adavid@clarkson.edu.