Fundamentals of Programming for Science and Engineering

Valerie Maxville
Curtin University
Perth, Australia
v.maxville@curtin.edu.au

ABSTRACT

It is common for science and engineering courses to include one computing unit, usually in first year. In a newly-developed first-year unit, we have combined Python coding, Science and Engineering applications and research-oriented skills to help students understand how coding may be applied in their studies and research. Student responses have been positive, and the unit continues to evolve in response to student and faculty feedback. With increasing uptake in the unit, it is hoped that a wave of computational literacy will foster an increase in the application of computational techniques by undergraduate and postgraduate students.

CCS CONCEPTS

• **Software and its engineering → Object oriented development**;

KEYWORDS

Software engineering, computational science, data science, programming

ACM Reference Format:
Valerie Maxville. 2018. Fundamentals of Programming for Science and Engineering. In *SE4Science'18: SE4Science'18:IEEE/ACM International Workshop on Software Engineering for Science , June 2, 2018, Gothenburg, Sweden,* Randy Bilof (Ed.). ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/3194747.3194752

1 INTRODUCTION

Computers are a necessity in almost every aspect of science and engineering. While computers and applications have become more user-friendly, scientific research requires more than literacy in computing. A scientist who is confined to using existing programs will be limited in the type and scale of research they can undertake. To go beyond the currently available applications, those doing research in science and engineering need to be able to produce code and scripts for their own experiments. In some cases, a research team may be big enough to include trained programmers. If not, the scientist/engineer will need to learn to code. Wilson [11] states that most scientists are never taught how to write, test, debug, install and maintain software. They are given a generic introduction to programming and/or statistics, which is detached from the rest of their program and focuses on a specific language or tool. In a perfect world, there would be multiple instances of programming and software engineering instruction throughout undergraduate studies in science and engineering; however we need to take the opportunities we have and revisit with the Carpentries [11] or similar later on.

In 2016, the author was tasked with developing a new programming unit for Data Science students at Curtin University. The unit would also be offered to students in Science and Engineering, and beyond. This was a perfect opportunity to apply experience from managing and delivering training at the Pawsey Supercomputing Centre, and draw upon learnings from leaders in education and training in Computational Science [9, 10] and Software Carpentry [11]. The unit, *Fundamentals of Programming*, was initially formulated as yet another generic programming unit, this time in Python. With permission from the Data Science course committee, the brief was changed to develop and deliver a unit to take students beyond the fundamentals: targeting skills to help them with their Science, as well as teaching them about automation, reproducibility, quality and sharing code with their community. A key requirement was that examples relate to the diverse backgrounds and study programs of the students, providing both relevance and potential for future application in their courses and research.

We have now delivered the unit twice, working with both undergraduate and postgraduate students in each semester. There has been a very strong positive response from students and faculty. In the following sections, the approach to the unit, the results and the future for the unit will be discussed.

2 APPROACH

Fundamentals of Programming (FOP) was developed as part of the multi-faculty Data Science undergraduate degree at Curtin University. This provided a starting point of a prepared handbook entry and assessment structure for the unit. Implementation of the unit was overseen by the Data Science course committee with representatives from across all faculties. With the target students coming from Data Science and the broader science courses, there was potential for the unit to generate students willing to try computational and analytical approaches to science.

2.1 Content

A review of relevant courses was compiled to ensure coverage of the most important concepts. Keys resources included Software Carpentry [2], Data Carpentry [1], Computational Science [9], Computational Thinking [10], the Computing Body of Knowledge, Skills Framework for the Information Age (SFIA, [5]), Data Science texts, Python courses [3, 8] and the author's experience and ideas. Of

SE4Science'18, June 2, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5748-7/18/06…$15.00
https://doi.org/10.1145/3194747.3194752

prime importance was aligning the content of the unit to the Computer Science unit *Object-Oriented Program Design* (Java) as they would become alternative prerequisites to later units: *Database Systems* and *Data Structures and Algorithms*.

Having distilled this review into a top level wish-list of topics, the task was to find an ordering suited to constructing the required knowledge (see Table 1). As some concepts take a few revisits to gel, a spiral approach was used to introduce, then reinforce and extend topics week on week. During planning, topics were arranged in swim lanes to reflect the area of computing they drew on to ensure continuity and reduce context-switching. As the prime focus of the unit is programming, a decision was made to begin with six weeks of coding standalone programs and scripts. These programs would include simulations and data processing, and scripting to allow automation of workflows and experiments. As the unit went on, more consideration would be made towards the research community and sharing code and data. Once the overall storyline was decided, the objectives for each week were defined, along with the key practical examples from a range of science domains. With that structure, the lecture content was developed to provide the students with the required background knowledge and motivation (examples and applications) to meet the objectives and implement the tasks.

The order of the topics is intended to force a focus on programming and being able to code in a very common, but not initially helpful environment (Linux/Vim). It is important for students to understand the automation potential available at the command line, and the easy workflows available with piping and redirection. Parameter sweeps are a particularly illustrative example - applicable in many areas of science and engineering, and are revisited in the assignment to reinforce the concept. After the first six weeks, students are experienced enough in that environment to have a greater appreciation of Jupyter notebooks. These are introduced as agents of reproducible research, and most students choose to use them in their assignments.

As mentioned previously, a spiral approach was taken to introducing and reinforcing content, where possible. Thus concepts such as sequences and slicing were demonstrated with strings, then lists, then arrays and multidimensional arrays over successive weeks to revisit the concepts and similarities.

2.2 Implementation

An early constraint was to use Unix/Linux as the learning environment. This prepares students for data science at scale, and helps them to develop a level of abstraction from the operating systems they are used to.

Each week students attend a 2-hour lecture, which is also recorded for students to revisit. There is also a 2-hour practical session with a maximum of 18 students in each computer laboratory. As an example, Learning Outcomes for Week 1 (Introduction) are:

(1) Understand and discuss the role of computers in science
(2) Understand and discuss basic computing concepts and the purpose of programming languages
(3) Define and use basic datatypes and control structures in Python
(4) Define and use key commands in the Linux operating system
(5) Use Python in a Linux environment to model simple systems

The assessment breakdown for the unit is: practical test 15%, worksheet test 15%, assignment 20%, exam 50%. The practical tests for FOP have been split into five small tests worth 3% each. Each tests aims to assess the competencies that we require students to have by the end of the unit. The expectation is that most students will achieve 100% in all of the tests. Another aim is to make the tests less about stress and more about learning. To that end, the tests are open book, open computer, with no time limit and the students can ask questions and get help. If they do not know how to complete the tasks at the start of the tests, they certainly do by the end. Informal feedback on the practical tests has been very positive in both semesters the unit has run.

Good software practices are introduced and enforced, while trying to avoid overloading them in their first coding unit. For each submission a markdown README file is required, listing the overall purpose, each program and a short description, dependencies and updates. Reproducibility is a key point throughout, including keeping track of versions of software/packages/environments used. Building from functions to modules to packages, students are shown how to select, develop and share packages with the community. The risks of using outside software is discussed, with tips on how to reduce risks by choosing well-supported, current packages. Coding style aligns to PEP-8 [4], and the mantra "readability counts" is reinforced regularly across the semester. Another software engineering aspect is the use of tools to assess quality - adherence to PEP-8, pylint and debugging tools were all discussed and explored in the practicals.

3 RESULTS

We will consider the results seen in the unit in terms of uptake and the student experience.

3.1 Uptake

When developing the unit for a Data Science audience, it was expected that there might be 20-40 students. The first semester we attracted in 120 students, Our numbers have increased each semester, with 170 students in Semester 1, 2018. The demographics of the students are very different to the Computer Science OOPD (Java) unit (2).

Students in the sciences have a choice of three programming units: Computer Skills (Scientific Data Information Management with Microsoft Office), Scientific Computing (Programming with Matlab) and Fundamentals of Programming. This semester, 30 students took Computer Skills, 110 enrolled in Scientific Computing and 170 in Fundamentals. In comparison, we have 370 students in OOPD (Java) and 300 in Engineering Computing.

Also of interest is the gender balance in FOP - 45% of undergraduate students are female. This compares very favourably with 5% in OOPD. The diversity of FOP students is high, however it is viewed. As such, it is important to use a variety of examples with (initially) high scaffolding to make the unit as accessible as possible.

3.2 Student Experience

Across the cohort, there have exciting journeys from students from strong and not so strong computing backgrounds. It is a challenging unit - no one gets through easily. The assignments are particularly

Table 1: Weekly topics in Fundamentals of Programming

Weekly topics	Subtopics/Additional Topics	Examples
Introduction	Computational Science, Python and Linux intro	Unconstrained growth
Strings and Lists	Random package	Monte Carlo
Arrays and Plotting	Numpy, matplotlib packages	Drug dosages
Multi-dimensional Arrays	Numpy, Scipy packages	Heat diffusion
Files and Grids	CSV files	Fireplan, Game of Life
Scripts and Automation	Parameter sweeps in bash	Drug dosages, weather plots
Data Wrangling	Regular expressions, tuples, sets, dictionaries	Data cleansing
Exploring Structured Data	Pandas package, reproducible research, Jupyter notebook	Text processing, Data Carpentry
Presenting Structured Data	Pandas, Seaborn, Bokeh packages	GovHack
Modeling the World with Objects	Object-oriented approach and definitions	Animal tracking, Bank accounts
Objects and Exception Handling	Developing and sharing packages	
Revision and Beyond	Databases, Games and GUIs, Web scraping	

Table 2: Student Uptake

Unit	Numbers	Female:Male Ratio	Main Disciplines
FOP undergrad	129	45:55	Multi-disciplinary Science, Actuarial Science, Data Science
FOP postgrad	42	14:86	Predictive Analytics, Electrical Engineering
OOPD undergrad	370	5:95	Computer Science

rewarding to assess. Both semesters have included a parameter sweep-driven investigation of hypothetical drugs, reinforcing the power of the approach. In addition, in Semester 1, students worked with data from the Bureau of Meteorology (BOM) [6] to investigate sustainability potential and issues in a chosen location, specifically renewable energy and water scarcity. Their reports were able to indicate not only the suitability, but also estimates of energy generation from solar and wind power. A few students are now looking into solar power for their homes, based on what they discovered in the assignment. The Semester 2 assignment looked at the newly released Australian Census data, which garnered a lot of negative attention due to cyberattacks on the Census website [7]. Students were tasked with defining three hypotheses each using at least three data items and three locations. They could also link data from other repositories. Some created informative maps of population, income and age, others compared the prevalence of indigenous languages in different parts of the state, patterns in religion, family units and vehicle ownership. To encourage planning and reflection on the assignment, the marking allocation was 25% justification of methodology (how they processed the data), 50% implementation (code, data presentation) and 25% interpretation of the results. Many students the marker on their journey of discovery, fully documented through Jupyter notebooks.

Feedback from students has been very positive and constructive. Student evaluations of the unit through eVALUate support the approach taken and its relevance to their studies.

"I just wanted to say that FOP was one of the best university coursework experiences I've had! I benefited from the well presented, structured learning content and nice awareness of the students perspective. Quite a contrast to my undergraduate experience at the <deleted> mechanical engineering school." (Email from student, FOP, S2 2017)

The practical tests were popular, as was the comparatively relaxed atmosphere:

"having the practice tests every few weeks during the workshops helped to implement and highlight areas that I wasn't familiar with or those that I was. having these tests open book also removed the high levels of stress associated with tests which I found very useful " (eVALUate, FOP, S1 2017)

FOP is a learning experience on many levels. It is often part of students' first semester of University, where they are transitioning from high-school. Much of that change can be daunting. It is important to provide a safe environment, where students see reward for effort, assessments are relevant, support is available and the unit is transparent with no surprises.

"I found FOP to be a well paced course, and easily accessible to those of us with little to no prior programming experience. The lectures were set out in an easy to understand format, and for the most part the pracs reinforced what we had learnt in the prior lecture which I found to be very helpful. I also liked that the practical tests were split up instead of being one huge test because it took a lot of the stress away from the test and allowed me to focus more on doing the test than worrying. The mid-semester exam was also a good test of our knowledge." (eVALUate, FOP, S1 2017)

The most important outcome is that students feel they can go further with programming, and can see its application in their field. One student described how he took an example from one lecture, and applied and extended it in his research assistant role that same week.

"For someone who really doesn't know much about programming this unit really shows you the power behind having the skill tucked under your belt. The pracs are always interesting and you're always learning new things, reinforcing things that were touched on in the

lecture, or witnessing your mistakes first hand and having to correct yourself. I feel like I can now say that I have a firm grasp on programming and know enough of the basics to be able to just search around and experiment with code to figure out what needs to be done. Thank you for this unit!" (eVALUate, FOP, S1 2017)

There has been some negative feedback about the unit. A number of students questioned the use of written tests and exams in a very practical unit. At this point we haven't found an alternative to hold large-scale simultaneous assessments. One student expected each lecture to be a code-along environment. We do have some live-coding in the lectures, however, unless all or most students can code-along, aligning to that approach would split the class. Overall, most of the issues have stemmed from students not understanding what to expect from the unit, and perhaps underestimating how difficult it would be.

4 CHALLENGES AND FUTURE

There have been a number of challenges in the delivery of the unit, and its integration with subsequent units:

- Diverse student backgrounds and expectations
- Finding tutors
- Preparing students for later units
- Increasing student numbers

As noted in Section 3.1, we have a wide range of students in FOP, with differing levels of interest and experience in computing, and studying a wide range of courses. Quite often our practical sessions run at two (or three) speeds: the experienced students rushing ahead, while those new to coding may struggle. Most of the students complete the assigned tasks within the allocated time. For example, with the practical tests, some students finish in 10-15 minutes, others take an hour, and some need two hours. By providing a range of extension materials and review questions, we give the more advanced students additional challenges to explore. We also encourage students to help each other - as both sides will reap benefits.

In our department, tutors are selected from our student-base, once they have completed 1-2 years of study. This ensures that the tutors know the units and where they lead. In the case of FOP, the unit is not available to Computer Science students. We are gradually locating strong students with an interest in Python. Exacerbating the problem, we have moved to two tutors per class of 18 students. This has been highly beneficial for providing timely support for students, but has stretched our resources.

Semester 1, 2018 has been our first cohort of students studying Data Structures and Algorithms in Python. We have 5 Python students in a class of 90 mostly Java students. Although FOP has been aligned to the Java unit (OOPD), there have still been some worried students. To support them, we have scheduled addition tutorial sessions to give more time and support to the students. There are some challenges working with languages as different as Python and Java for implementing data structures. As an example, the Java students have only worked with arrays, while the Python students start with lists. Then we teach them array-based stacks and queues on the way to linked lists and dynamic structures. Probably more of an issue is increasing the amount of testing to match the level of coverage in OOPD. As a result of the students' experience, and reflection on FOP, we will be adding more testing throughout the unit and a new lecture and practical will focus on testing and exceptions.

This year the Bachelor of Technology students moved to FOP as their core first year unit, instead of OOPD using Java, We have also discussed Geophysics taking up the unit, rather than developing their own course. Perhaps the most significant change to come is the decision by the Engineering steering committee to move from C to Python/FOP for their common first year. With this move, the students will replace a 12.5 credit unit with the 25 credit FOP - a marked change to their program. We will be working to nurture additional tutors in preparation for the unit to increase by over 100 students next year.

5 CONCLUSIONS

The first year of running this unit has been very positive, with both students and faculty showing support and moving towards this approach to teaching programming. We have been able to straddle the multiple requirements for the unit, while staying true to computer science and software engineering principles. The approach to date has been different to how we would typically teach Computer Science, and student feedback indicates that they appreciate that approach. Key modifications in the current semester are to have an increased focus on testing, reduce the structured data coverage to one week while adding another week of object orientation and exceptions. We are also looking into what alternatives are available to using written tests/exams. The competency-based practical tests have been very successful and will continue.

The unit is a work in progress, and will continue to evolve and become more inclusive of the range of disciplines our students are studying. We feel it has been a success so far: exposing students to good coding practice and creating awareness of the possibilities of computational and data science.

REFERENCES

[1] Data Carpentry. 2018. Data Carpentry. (Feb. 2018). Retrieved February 2, 2018 from http://www.datacarpentry.org/

[2] Software Carpentry. 2018. Software Carpentry. (Feb. 2018). Retrieved February 2, 2018 from https://software-carpentry.org/

[3] Allen B. Downey. 2018. Think Python: How to Think Like a Computer Scientist. (Feb. 2018). Retrieved February 2, 2018 from http://greenteapress.com/thinkpython2/html/index.html

[4] Python Software Foundation. 2018. PEP 8 – Style Guide for Python Code. (Feb. 2018). Retrieved February 2, 2018 from https://www.python.org/dev/peps/pep-0008/

[5] SFIA Foundation. 2018. Shodor, a national resource for computational science education. (Feb. 2018). Retrieved February 2, 2018 from https://www.sfia-online.org/en

[6] Bureau of Meteorology. 2018. Climate Data Online. (Feb. 2018). Retrieved February 2, 2018 from http://www.bom.gov.au/climate/data/

[7] Australian Bureau of Statistics. 2018. Data by Region. (Feb. 2018). Retrieved February 2, 2018 from http://stat.abs.gov.au/itt/r.jsp?databyregion

[8] Zed A. Shaw. 2018. Learn Python the Hard Way. (Feb. 2018). Retrieved February 2, 2018 from https://learnpythonthehardway.org/

[9] G.W. Shiflet, A.B.; Shiflet. 2006. *Introduction to Computational Science : modeling and simulation for the sciences* (1st. ed.). Princeton University Press, Princeton, NJ.

[10] Shodor. 2018. Shodor, a national resource for computational science education. (Feb. 2018). Retrieved February 2, 2018 from http://www.shodor.org/

[11] G Wilson. 2014. Software Carpentry: lessons learned [version 1; referees: 3 approved]. *F1000Research* 3, 62 (2014). https://doi.org/10.12688/f1000research.3-62.v1