PGCAP Module 3 Summative Assignment

David Paredes
Atomic and Molecular Physics
Department of Physics
Durham University

1 Introduction

When a postgraduate student enters the department of Atomic and Molecular Physics, one of his first tasks is to attend the Graduate course, which will train him/her in the most basic skills and knowledge necessary to undergo the research they are presented with. This course aims to fill in the gaps, and most students will find particular areas in which their own training might surpass the content of the course. Still, this course is very important to bring the students up to speed, to obtain a *common ground* from which to start diversifying their own research.

In the course, students are asked to solve some problems which, in some cases, involve the use of a computer; however no formal training on computing is given within the course[1]. These tasks can range from analysing large sets of data, solving differential equations, plot solutions to certain equations, etc., and some of them might require the use of a computer. Little or no guidance is provided on how to undertake these, mainly to avoid constraining the students on a particular tool or method.

During their research they are confronted by similar problems but within a more complex context. Some students realise that they require learning how to program in order to solve some problems more efficiently However, learning how to program is a complicated task [1], and most students spend a substantial amount of time learning a programming language and understanding its capabilities.

We wanted to design a programming course, based on the Python language, to be used at the graduate level in the Atomic and Molecular Physics department. Python is a successful programming language: it is rapidly becoming one of the most popular introductory teaching languages in the world [2], partly due to its emphasis on *code readability*; it is consistently ranked amongst the most popular programming languages [3], and it is used by many large organisations [4]. It is a scripting language: that is, a programming language that allows the interpretation (contrary to compilation) of "scripts" that can also be run, alternatively, line-by-line in the console. One of the benefits of Python (and other scripting languages) is the possibility to use the console as the glass-box inside the black box [5]: i.e. a way to evaluate the state of the program during execution, and thus avoiding common problems arising from the lack of knowledge of the state of the system. This is in sharp contrast with other compiled

[1] Many Physics undergraduate programmes around Europe offer some training on programming. However, that training is not always thorough, and students are not always able to solve general problems using their programming language.

languages, which do not offer the possibility to debug in real time. Also, Python is *free* (contrary to some other proprietary computing packages), and so encouraging students to use it could reduce the costs associated with software licenses in the department.

The aim of the course is to lessen efforts that the students make independently to (re)learn a programming language, and to showcase the capabilities of the Python language in solving scientific problems directly related to their research, thus motivating students to use it as a general-purpose tool. The course is intended to provide learners with some context in which to use Python close to the discipline of Atomic and Molecular Physics. This is done in two ways: first, identifying situations in which students are likely to find programming useful; and second, by providing them with examples that they can directly re-use in their day-to-day work. There exist other courses which showcase the capabilities of Python as a scientific tool (see, for example, the Python Scientific Lecture Notes [6]). These courses, though highly encouraged as a reference to improve the technique, lack the specificity we required to motivate students to use Python in their research.

Figure 1: Logo of the course

This document reports the efforts required to implement the course mentioned above. The implementation of this project has been done in cooperation with Thomas Ogden and James Keaveney, researchers in the AtMol department. First, we talked to the members of the department to understand what their requirements and preferences were. Then we defined the course aims and educational outcomes. We also set down a set of guidelines to homogenise the different parts of the course. Finally, we started with the implementation of the course, which can be found in [7][2].

2 Survey

To organise this course, we required input regarding the contents of the course from the members of the department: postgraduate students, postdoctoral researchers and principal investigators (PI). To this end, we organised a survey in the department to understand if this course would gather sufficient audience, and what were the most interesting topics for the intended audience. After some informal conversations, we arrived at three key aspects that affected the characteristics of the course.

- Length of the course (short, intensive course; or spread during several weeks)
- Date of the course (during term-time or during the summer period)
- What topics would they find interesting/important

We sent an email to the department and received 19 responses to our email: 9 postgraduate students, 4 postdoctoral researchers and 6 PIs. The email also prompted informal conversations with other members in the group and outside of it. In the following

[2]Once the course is finished, the materials generated will be made public using the MIT license.

paragraphs, I summarise the answers from the two distinct target groups (postgraduate and postdoctoral researchers, and PIs).

2.1 Postgraduate students and postdoctoral researchers

All the correspondents in this category showed interest in the course. Some of this interest might be explained by considering that during the previous years, some of the members of the department have been showcasing Python and the possibilities that the language has.

In terms of length and date, the answers varied wildly, ranging from people who wanted a short, intensive course at any point in the year, to longer courses, with space between lectures, during term time. However, one of the answers showed an important constraint: if the course was done during the summer period, it might clash with people's vacations, and it should be avoided if possible.

Regarding the content of the course, the vast majority of the answers came from sporadic users of Python who wanted to use the language for particular applications, such as: data analysis, matrix operations, debugging / profiling / optimisation tools, interfacing Python with scientific instruments, how to transfer and store data in different formats. Some of these students (specially former Durham undergraduates) had learned Python as part of their undergraduate courses, but were not confident enough in the language to use it as an everyday tool.

Outside of the average requirements, shared by the majority of the postgraduate and postdoctoral researchers, there were some outliers, both at the basic and the advanced level. In the first group we found people who needed basic training on how to install the programming environment and who required some advise on the syntax. On the other end, some people wanted advanced features like GPU programming, multithreaded/multiprocessor capabilities of the language, remote data acquisition, and integration with other languages.

Also, in several occasions, people asked for advice on general concepts like good coding practices, documentation procedures and version control, both for single-user bookkeeping and to write programs collaboratively. They also wanted use cases, and information about common pitfalls and solutions.

2.2 Principal investigators

Contrary to the input obtained from postgraduate and postdoctoral researchers, mostly in email form, PIs wanted to meet in person to discuss different aspects of the course. This prompted interesting conversations, and the opportunity to discuss more in depth the course proposal[3].

Some of the principal investigators wanted answers to very specific questions: how to propagate the single-particle Schrödinger and nonlinear Gross-Pitaevskii equations, matrix problems, manipulation and diagonalisation, solve the optical Bloch equations, etc. ; however, most of them were interested in a very generic range of skills: data analysis, curve fitting, numerical solutions to equations, generation of publication-quality figures, re-usability, and writing efficient and clear code.

Conversations with some of the PIs suggested that the course should happen either immediately before or together with the graduate course, so that the students can use

[3]Some of the PIs were even interested in taking the course once it was developed

the acquired knowledge in solving the specific problems set out in the graduate course homework.

3 Design of the course

After reviewing the input received, we decided to orient the course of students who already had programming experience, and that were familiar with some of the basic features of the Python programming language. For those students who lacked this basic training, we suggested the use of the self-learning materials from the Durham University Physics Laboratory guide [8]. Students that previously took part in the Durham University Physics undergraduate programme would only require to review this knowledge, since they are trained in using Python in Levels 2 and 3 of their degrees[4].

Developing programming skills is a complex process that requires many high-level cognitive activities [1, Section 3.3]. Usually, novice learners of programming in the Physical science limit their learning to the acquisition of *surface* knowledge (as opposed to *deep learning* [9]) because they *lack detailed mental models* of both the domain of application and programming and fail to apply relevant knowledge [1]. A typical *threshold concept* [10] that appears early on during the learning process is "the notion of the system making sense of the program according to its own very rigid rules [which is] a crucial idea for a learner to grasp" [11], which is usually referred to as by the aphorism " A computer program does what you tell it to do, not what you want it to do."

Programs are usually written for a purpose. Literature suggests that students do not learn *holistically*, by adding general facts to their knowledge, but rather *opportunistically* [1], and programming can be regarded as"an incremental problem-solving process where strategy is determined by localized problem-solving episodes and frequent problem re-evaluation" [12]. Therefore, a course based on just technical, abstract aspects would be counterproductive, since students might not be able to use these technicalities to solve the problems they are faced with. Also, only a limited amount of learners (those with an abstract learning tendency [13, 14]) would benefit from the course.

Following this discourse, we structured the course as a set of lectures/use cases with direct applications in the student's research: students will find useful to have received formal training and programming experience in problems that they will have to re-visit later in their research life. Ideally, these classes should take place during 4 or 5 afternoon/morning sessions in the week before the graduate course is due to start. These sessions, each of them spanning 2 hours approximately, should take the form of relatively short lecture-like demonstration followed by problem solving activities. The initial part of the sessions would introduce the topic and some of the technical aspects, possibly in an abstract way. The problem solving part would focus on one or more problems related to the topic covered during the session. The course structure (mixing small amounts of lecturing with many use cases) benefits our target audience by contextualising programming.

From the input that we received, we identified four distinct topics where students might benefit from using Python:

- Input and output of data
- Data analysis

[4]For newer programmes, this has changed, and the Python courses take place on different Levels.

- Numerical methods
- Plotting

These answered the (very practical) questions: *how to get data from outside the program?*, *how to generate data?*, *how to process data?*, and *how to represent data?*. These questions frame Python as a general-purpose tool to solve problems in both theoretical and experimental research.

To integrate the initial demonstrating part in each session with the hands-on part of the class, we developed a series of IPython Notebooks [15]. The IPython notebook *"is a web-based interactive computational environment where you can combine code execution, text, mathematics, plots and rich media into a single document"*. This environment allowed us to prepare in advance the introductory discourse and some examples that can be executed, changed and visualised in real time. These notebooks can be shared, and can be executed locally (if the students have IPython Notebook installed), or they can be accessed using an online renderer, *nbviewer* [16]; it is also possible to export these notebooks as static webpages (to be visualised in a web-browser), or as standalone python scripts. Using these notebooks allows online access to all the materials of the course, which could be the basis for an open, self-taught course.

Each of the notebooks focused on a particular, technical problem (for example, a notebook under the topic "Plotting" focuses on constructing a complex, multi-panel plot, combining raw data with different fitting curves, and to show the residuals). Some of these notebooks come with questions and problems showing progressive difficulty at the end; these should be solvable by simple alterations of the code presented in the notebook. In the implementation of the course as part of the graduate course, the solution to these problems would constitute a form of formative assessment, which should be followed by feedback to the students.

At the end of each topic, a long, complex problem (called *quest*) was proposed. These quests present typical problems found during research in the field, and are redacted in an amenable/narrative way to provide with some "context" for each of the problems. For example, we can read in the quest of the Input/Output topic: *First week here: hopefully you have done all the bureaucracy required of you, and now you are ready to do some science. Ready or not, the postdoc you are working with is asking you to get the data they have been gathering all morning in the coincidences experiment and save it in a format that makes more sense. [...] Your task is to write a script that converts the data in the files provided to time tags, and save them in a file called "output.csv" ...*

These quests form the summative assessment of the course, and enough time would be given to the students to finish them.

Alongside with these four topics, we wrote some introductory material with general information about Python and ways to get help, as well as some information regarding the documentation and commenting of the code. Documenting (and commenting) code are important topics, as a suitable documentation can help both code re-usability and collaborative development [17, 18].

Regarding the impact of the course in women, which is an important topic in STEM subjects, more than 50% of the female members in the department answered the survey, which indicates that the women in our department were already motivated by the prospect of such a course. If we consider, however, the impact of the course outside of the department (when it becomes open to the public), we note that the course structure (mixing small amounts of abstract lecturing, with use cases) has the potential to attract the

female audience by appealing to the *concreteness* of its examples: there are studies that suggest that women learners value putting computing studies in context [19, Chapter 3]; these studies refer to a broader context than that of computing, and the applications the field has in the "real" world [20]. However, there is also considerable evidence suggesting a discrepancy between male and female learning preferences [21], especially regarding the abstract-concrete dimension of learning (in the framework of Kolb's learning styles [13, 14]): studies suggest that women score higher in the concrete side of the continuum, whereas men tend to score higher in the abstract end.

4 Further work and conclusions

Some of the PIs suggested that the demonstrators acted as experts in solving a real-world problem. This should demonstrate what the thought process of an expert user of Python is, giving hints to the students about the important aspects of the problems they will be faced with. However, one needs to bear in mind that learning by observing an expert is of limited effectiveness in most contexts [22], so other alternatives should be explored. We can imagine the expert as guide in a tutorial-like setting for a relatively long time-span, in which students solve collaboratively a complex problem using Python. If more time was available, it is an avenue that could be explored further.

In conclusion, we have designed a course which aims to prepare students to use Python as a general-purpose programming language in the research concerning Atomic and Molecular Physics. First, we surveyed the department to understand what the requirements and preferences were. Then, we designed a course based on these requirements that was divided into four distinct topics identified as representative of the kind of research carried out in the department: data input and output, data analysis, numerical methods and plotting. We used IPython notebooks as a tool to deliver both lecture-like content and hands-on practice to work on some use cases.

5 Appendix: "Intro to Python for Graduate Students" course description

5.1 Course aims:

"Intro to Python for graduate students" is a week-long course designed to introduce the casual Python user to the capabilities of the language in the context of scientific computing at the graduate level.

It aims to prepare students to use Python as a general-purpose programming language in research, to be used both in theoretical and experimental contexts. Tasks like data taking/handling/analysis, generating publication-quality plots, solving differential equations,... will be covered, following the standards in the scientific community.

It consists of self-contained topics that focus on different tasks: Input/Output, plotting, data analysis and scientific computation. They can be coursed separately if required, according to the needs of each particular student. The course is delivered as a series of worked examples that can be used online or as part of the graduate course in Atomic and Molecular Physics. An experienced user will guide the students through the sections and the examples in morning/afternoon long tutorials, to help the students in understanding the tasks at hand, and support them in the usage of Python.

5.2 Learning outcomes:

By the end of the course, the student should be able to

- know how to install and use the Python environment in their preferred OS
- know where to get information about the programming language, and how to navigate on- and offline manuals;
- identify and produce programs following standards for commenting and style,
- perform simple tasks using Python to help them in their daily scientific routines,
- use Python to generate publication quality plots, with multiple subplots and insets,
- use Python's numerical packages numpy and scipy to perform numerical calculations (linear algebra, differential equations, ...),
- recognise possibilities for simple optimisation in their scripts and ways to implement them,
- get information about the more advanced packages written for Python to solve quantum mechanical problems (parallelisation, QuTIP,...).

References

[1] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[2] Python is now the most popular introductory teaching language at top U.S. Universities | blog@CACM | Communications of the ACM. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext.

[3] TIOBE Software: Tiobe Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[4] Python Success Stories — Python.org. https://www.python.org/about/success/.

[5] Benedict Du Boulay, Tim O'Shea, and John Monk. The black box inside the glass box: presenting computing concepts to novices. *International Journal of Human-Computer Studies*, 51(2):265 – 277, 1999.

[6] Python Scientific Lecture Notes Scipy lecture notes. https://scipy-lectures.github.io/.

[7] Intro to Python for graduate students. http://nbviewer.ipython.org/url/www.ambages.es/pythonCourse/python_notebooks/Python%20Course%20-%20Overview%20Page.ipynb.

[8] Python Resources, Durham University Physics Laboratory Guide. http://labs.physics.dur.ac.uk/computing/resources/python.php.

[9] J Biggs. Aligning teaching for constructive learning. Higher Education Academy online resource. Last accessed 1/10/12.

[10] Jan H. F. Meyer and Ray Land. Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *Higher Education*, 49(3):373–388, April 2005.

[11] Benedict Du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[12] Simon P. Davies. Models and theories of programming strategy. *Int. J. Man-Mach. Stud.*, 39(2):237–267, August 1993.

[13] D.A. Kolb. *Experiential learning: experience as the source of learning and development*. Prentice Hall, Englewood Cliffs, NJ, 1984.

[14] Higher Education Academy. Learning Styles. http://84.22.166.132/learning-and-teaching-theory-guide/learning-styles.html. Accessed 1/10/2013.

[15] The IPython notebook. http://ipython.org/notebook.html.

[16] nbviewer - a simple way to share IPython notebooks. http://nbviewer.ipython.org/.

[17] Lutz Prechelt, Barbara Unger, and Michael Philippsen. Documenting design patterns in code eases program maintenance. In *In Proc. ICSE Workshop on Process Modeling and Empirical Studies of Software Evolution*, pages 72–76, 1997.

[18] Dani Nordin. Documenting for end users and the production team. In *The Definitive Guide to Drupal 7*, pages 221–226. Apress, January 2011.

[19] Jane Margolis and Allan Fisher. *Unlocking the Clubhouse: Women in Computing*. MIT Press, 2003.

[20] Jill Dimond and Mark Guzdial. More than paradoxes to offer: Exploring motivations to attract women to computing. *Georgia Institute of Technology Technical Report*, 2008.

[21] Sadan Kulturel-Konak, Mary Lou D'Allegro, and Sarah Dickinson. Review of gender differences in learning styles: Suggestions for STEM education. *Contemporary Issues in Education Research (CIER)*, 4(3):9–18, March 2011.

[22] M. T. Chi. Learning from observing an experts demonstration, explanations, and dialogues. *Expertise and skill acquisition: The impact of William G. Chase*, pages 1–28, 2013.