Jupyter Notebooks as Computational Thinking Tools for Teaching and Lifelong Learning in Biotechnology

R. Lebrón[1](✉), A. Ortiz-Atienza[1], S. Bretones[1], C. Capel[1], F. J. Yuste-Lisbona[1], and A. Cámara-Artigas[2]

[1] Center for Research in Agri-Food Biotechnology (CIAIMBITAL), University of Almeria, 04120 Almería, Spain
rlebron@ual.es

[2] Dept. Chemistry and Physics, ceiA3, University of Almeria, 04120 Almería, Spain

Abstract. To remain operational during the COVID-19 pandemic, educational institutions had to rapidly migrate to online learning, affecting 1.6 billion students worldwide. This unexpected scenario required extra effort for teachers and students, but it also provided an opportunity to encourage students to develop new skills, such as computational thinking, and to foster lifelong and autonomous learning. Computational thinking is not only one of the cornerstones of programming, but scientists from all disciplines use it to solve problems even when they are not programming. Literate programming is a paradigm that facilitates the acquisition of these skills by combining explanatory texts in natural language with executable code snippets and their output to familiarize a broad public with how code works and the reasoning behind it. These tools are known as interactive notebooks, with the Jupyter Notebook platform standing out for its simplicity, support for multiple programming languages, and ability to be used online. In this paper, we share our experience with Jupyter notebooks for teaching biotechnology, highlighting the pedagogical patterns we have found most useful to foster computational thinking among students with no prior programming background and providing tips to help extend the use of these tools in teaching.

Research Contribution: In fast-paced, ever-changing research and industries, the ability to face new challenges is essential. To assist biotechnology students in becoming problem solvers and critical thinkers, we propose an approach based on the development of computational thinking skills using interactive notebooks as training materials.

Keywords: Literate programming · Computational thinking · Lifelong learning · Online teaching · Biotechnology

1 Introduction

The onset of the COVID-19 pandemic in 2020 meant the disruption of educational systems in virtually every country, as the rapid spread of SARS-CoV-2 forced the decision

L. Daadaoui et al. (Eds.): ICESD 2022, AHSSEH 5, pp. 167–175, 2023.
https://doi.org/10.2991/978-2-38476-036-7_16

to close universities and schools, affecting nearly 1.6 billion students in over 190 countries (i.e., 94% of the world's students) [1]. This unexpected situation resulted in a quick migration to online education via various platforms such as Blackboard (https://www.blackboard.com/), Google Classroom (https://classroom.google.com/), Microsoft Teams (https://www.microsoft.com/microsoft-teams/), Canvas (https://www.instructure.com/), and VEDAMO (https://www.vedamo.com/). Adapting to this new educational environment was a challenge for educational institutions [2], but also provided a chance to encourage students to gain new abilities, such as computational thinking [3], and to foster lifelong and autonomous learning [4].

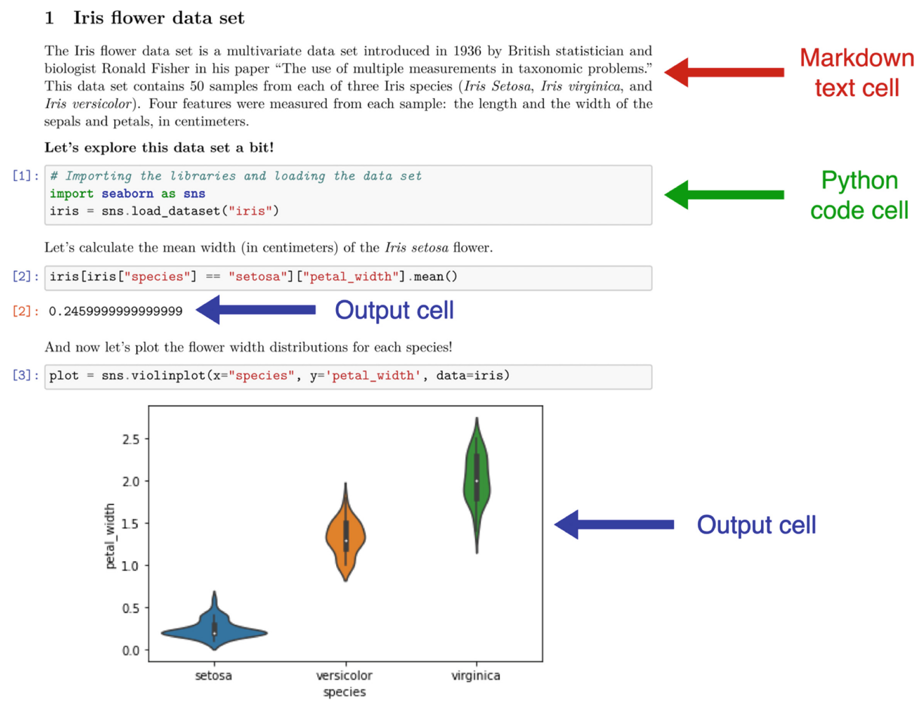
Computational thinking has been highlighted as a set of problem-solving skills that everyone should learn and use [5], and some non-profit organizations, such as the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA), promote its use in K-12 education [6]. These skills are founded on the following principles [7]:

- *Abstraction*: representing only the essential aspects of reality that are required to solve a problem.
- *Pattern recognition*: identifying and generalizing different cases.
- *Decomposition*: breaking down problems into simpler ones that can be solved separately.
- *Algorithm design*: defining each task as a set of step-by-step instructions.

Although these skills are tightly linked to programming, having a programming background is not required to put them into practice, and they should be understood as a cognitive process that can benefit virtually any human activity [8]. Computational thinking has been supporting the development of molecular biology for decades, with the discovery of the genetic code in the 1960s [9] likely being one of the first milestones involving it, because the step-by-step explanation of how the sequence of a mature messenger RNA is translated into a protein sequence is, in fact, an algorithm. Of course, computational thinking is one of the foundations around which programming is built and is an excellent spot to start learning it. Furthermore, it is preferable to have tools that bring the reasoning behind programming closer to the broad public to facilitate the learning of computational thinking. We believe that literate programming tools can meet this need.

Literate programming is a paradigm that aims to make program code more understandable by providing detailed explanations of how it works. Literate programming tools are used to generate what are known as interactive notebooks, which interleave explanatory text in natural language (often accompanied by mathematical notation, images, audio, or video) with executable code snippets and their output, which may include text, images, or tables among others (Fig. 1) [10]. Jupyter Notebook (https://jupyter.org/) is the most extensively used literate programming tool among programmers and scientists worldwide, and it is renowned for supporting many prominent programming languages, including Python (https://www.python.org/), R (https://www.r-project.org/), Julia (https://julialang.org/), JavaScript (https://www.javascript.com/), and C/C++ (https://isocpp.org/) [11]. The explanatory text can be enriched by using different formats and structuring it in sections thanks to the use of the Markdown lightweight markup language (https://daringfireball.net/projects/markdown/). It can be written in any human language that can be transcribed into one of the 161 writing systems currently supported by the Unicode encoding standard (https://unicode.org/), and it can even include emoji. In addition, code snippets written in the LaTeX document markup language (https://www.latex-project.org/), such as formulas and vector graphics, can be included in the explanatory text. Users can access Jupyter notebooks online without installing anything on their computers by using a modern web browser and regardless of the operating system, thanks to platforms such as JupyterHub (https://jupyter.org/hub) and Binder (https://mybinder.org/). Furthermore, a modified version of Jupyter Notebook called Google Colab (https://colab.research.google.com/), which is available online for free, allows users to access files hosted in their Google Drive (https://drive.google.com/) accounts and perform version control of their notebooks through GitHub (https://github.com/). Since it is a cloud computing service, Google Colab is probably one of the most suitable options for teaching, as educational institutions do not need to have their own Jupyter servers (e.g., their own JupyterHub servers). In addition, the loading times of this platform are much shorter than those of Binder (which is also a cloud computing service), and it has the extra advantage of being able to store both notebooks and input and output files in Google Drive.

As biotechnology teachers and researchers, we are aware of the rapid growth of our field in recent decades, as well as the fact that no single curriculum can cover all the current or future skills in demand in industry, and we strive to help students become lifelong learners, critical thinkers, and problem solvers. Our students do not take any courses with significant computational content until the bioinformatics subject in the third year of their biotechnology degree, and most students take this course with no prior programming knowledge. For this reason, we have to make an extra effort to help our students develop mathematical and computational skills, which are increasingly in demand in both research and industry but which have historically been neglected in biotechnology education [12]. We found Jupyter notebooks to be useful tools for teaching computational thinking skills and have developed our own notebooks as well as adapted third-party notebooks to enrich our students' training in mathematical and computational skills applied to the field of biotechnology. Some examples include our in-house notebook for detecting DNA mutations that cause phenotypes of interest using mapping-by-sequencing methodology [13] as well as our simplified and adapted version of the third-party notebook ColabFold [14] for predicting the 3D structure of proteins using deep learning models based on their amino acid sequence. We designed these notebooks to be self-contained, so that students always have explanations of key concepts, objectives, and methodology at their disposal, and so that the notebooks can be used in both face-to-face and online sessions, as well as for autonomous learning. Using Jupyter notebooks is also a good opportunity to introduce our students to both Python programming (an easy-to-learn, high-level, general-purpose programming language) and lifelong learning, as there is a wealth of free educational resources that are available in this format.



1 Iris flower data set

The Iris flower data set is a multivariate data set introduced in 1936 by British statistician and biologist Ronald Fisher in his paper "The use of multiple measurements in taxonomic problems." This data set contains 50 samples from each of three Iris species (*Iris Setosa*, *Iris virginica*, and *Iris versicolor*). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Let's explore this data set a bit!

```
[1]: # Importing the libraries and loading the data set
     import seaborn as sns
     iris = sns.load_dataset("iris")
```

Let's calculate the mean width (in centimeters) of the *Iris setosa* flower.

```
[2]: iris[iris["species"] == "setosa"]["petal_width"].mean()

[2]: 0.24599999999999999
```

And now let's plot the flower width distributions for each species!

```
[3]: plot = sns.violinplot(x="species", y='petal_width', data=iris)
```

Fig. 1. A notebook example with text, code, and output cells.

Here, we share our experience with Jupyter notebooks for teaching biotechnology, indicating the pedagogical patterns we have found most helpful in promoting active learning of computational thinking among our students with no prior programming knowledge, and providing a collection of tips to assist other teachers who want to incorporate these tools into their lessons.

2 Method

If properly designed, Jupyter notebooks are an excellent companion to lectures and can replace traditional educational materials, such as static textbooks and slides. Each Jupyter notebook we use for teaching biotechnology is self-explanatory, limited to a single topic related to previous lectures and designed to put theoretical concepts into practice. Most of our students have no prior programming experience, but code is an effective formalism for expressing problem-solving strategies that are based on computational thinking, so we expose our students to code written in Python and teach them how to read it, understand it, and express its meaning in their own words. To facilitate the learning process, our notebooks are designed using different pedagogical patterns [15] that have previously been described by Barba et al. [16]:

- *Shift-Enter for the win.* This is the most passive approach, but it serves as a preparation for the other approaches. It is based on using the notebook as a textbook. The teacher

runs from top to bottom of the notebook, executing each cell (by pressing shift + enter) at the same time as the students, while explaining the code, the abstractions behind it, and the interpretation of the results. The experience can be greatly enriched by using widgets that allow students to modify parameters to alter the results.
- *Tweak, twiddle, and frob.* We give students a complete notebook and ask them to change specific aspects of the code and interpret the changes in the results. We want students to understand how the code works and the relationship between parameters, input data, and results.
- *Fill in the blanks.* We give students a complete notebook with only a few small gaps to fill in. These gaps are accompanied by hints in the form of code comments, and the notebook includes tests to allow students to confirm whether their solutions are correct. Our aim is to get students to focus on a specific aspect of the workflow.
- *Bug hunt.* We give students a complete notebook with deliberate errors that they must find and correct. They are not given any hints, but the notebook includes tests to allow students to confirm whether their solutions are correct. Our aim is for students to gain a better understanding of how the workflow as a whole works.

We chose these four pedagogical patterns because they are easy to implement both in real-time sessions and for autonomous work, individually or in groups. Moreover, despite their simplicity, these pedagogical patterns allow a wide range of exercises, such as reading the code to try to understand how it works in general terms, examining specific aspects of its design, or editing it to modify its operation. Due to its more introductory and explanatory nature, we typically reserve the *Shift-Enter for the win* pedagogical pattern for real-time face-to-face or online sessions, though it is also appropriate for students to independently review the subject and key ideas before and after the lesson. Regarding the other three pedagogical patterns, we use them for both classroom exercises and homework.

3 Findings and Discussion

Jupyter notebooks are a new type of teaching material that stands out for its interactivity and for being friendly for both the designers (i.e., teachers) and the users (i.e., students). Furthermore, our students have found that using these notebooks is a handy way to take notes and gather theoretical concepts, exercises, questions, and answers into a single document. Jupyter notebooks are suitable for face-to-face, online, and autonomous learning, but good practices must be followed to reap the benefits. Both the guide to teaching and learning with Jupyter by Barba et al. [16] and the large-scale study on the quality and reproducibility of Jupyter notebooks by Pimentel et al. [17] discuss good and bad practices as well as anecdotes about using these tools in teaching and research. Based on these previous works and our extensive experience using Jupyter notebooks in biotechnology education, we have compiled a list of ten simple tips for making the most of these tools and avoiding the most frequent issues:

i. *Take care with the notebook's title, structure, and length.* Jupyter notebooks are IPYNB files. Unfortunately, the notebook's name is always the same as the file's name, which can cause issues if the name is too long or contains certain characters. For the IPYNB file, we recommend using a short name consisting only of alphanumeric characters and underscores and indicating the full name of the notebook in the first cell using a Markdown level 1 heading (e.g., # Making Phylogenetic Trees in Python). We recommend using higher-level headings to define sections and subsections to structure the document. Try not to let the depth of the explanations decline throughout the document, as the final section of Jupyter notebooks tends to be the least elaborate [17]. Regarding document length, we recommend creating short, self-explanatory documents dealing with single topics. It is preferable to have several short, related notebooks within the same directory than a single long, confusing notebook.

ii. *For each meaningful unit, use a single code cell.* If several lines of code work together to complete a task, group them in the same cell; otherwise, the students may encounter execution errors. Also, avoid grouping lines of code that perform different tasks in the same cell, as this may make it difficult for the students to distinguish between each step of the algorithm. It is also critical to use Markdown text cells before and/or after each code cell to explain how it works and how to interpret its output. We recommend that comments within the code (i.e., comments specific to the programming language being used) be reserved for hints to the students on exercises that involve completing or correcting code.

iii. *Check that the proper version of all dependencies is installed.* It is critical that you ensure that all the dependencies required by the students are installed in the environment in which they will be working (e.g., Google Colab) and that the version is the same as the one used to prepare the materials. Otherwise, include the code or instructions for installing the dependencies at the very beginning of the notebook (before importing them). We also recommend importing all required dependencies at the beginning of the notebook, before the rest of the code cells (and after the installation cells, if any).

iv. *Examine the input files thoroughly.* Ensure that all files required by students are available online (via Google Drive shared folders or otherwise) and check their completeness, formatting, and content. An incomplete or improperly formatted file may cause confusion among students, reducing their engagement during the rest of the session.

v. *Before the lecture, rerun the notebook from beginning to end.* This will assist you in detecting both accidental errors, such as those caused by out-of-order or deleted cells, and discrepancies between the environment in which the materials were developed and the environment used by the students. Perform these checks ahead of time, and then adapt or correct the notebook before sharing it with students.

vi. *Check that your students understand key concepts and have feedback strategies.* We recommend spending a few minutes in the beginning and/or before the conclusion of each session reviewing the most important aspects, such as subject-specific or programming concepts, Python or Markdown syntax, or the use of Jupyter itself. Remember that your students are just getting started as problem solvers. It is also critical to prepare in advance for feedback strategies that are appropriate for the teaching modality (face-to-face, online, or autonomous). For example, in face-to-face sessions, students can be given sticky notes with different colors and meanings (green: I finished the exercise; yellow: I am working on it; red: I got stuck), which they can stick to their desks or the back of their computers. As a result, the teacher can quickly assess the current situation of the class and decide how to proceed. In the case of online sessions, students can be asked to copy and paste their proposed solutions into private chats or share their notebooks with the teacher after they have had a reasonable amount of time to solve the exercise. In the case of an autonomous learning notebook, it is advisable to include self-assessment activities as well as a solved version of the notebook that students can refer to if they get stuck.

vii. *To explain abstract concepts, use metaphors.* Metaphors are figures of speech that help explain ideas or make comparisons by using representations of fictitious objects or actions that somehow resemble the abstract objects or actions you are trying to explain to your students. They have been shown to be helpful for teaching computational thinking [18] and programming [19], but should be used with care to avoid misunderstandings [20].

viii. *Create functions and object classes to practice abstraction.* When possible, break problems down into simpler tasks and explain each of these tasks to your students as if they were a separate problem. Using whatever resources that you deem necessary, explain the abstract concepts underlying each task (e.g., drawing the process often helps). Implement an example of this task step-by-step, then try to derive other examples by modifying the code. To show how to automate this task, abstract functions from the code examples and define object classes with appropriate attributes representing the function's input and output. It is sometimes interesting to represent a task as an action that an object can perform (i.e., a method of the object).

ix. *Experiment with various pedagogical patterns until you find the best one for each case.* We have indicated those pedagogical patterns that we use most frequently with our biotechnology students, but we strongly recommend you look at the catalogue of pedagogical patterns elaborated by Barba *et al.* [16].

x. *Share and reuse notebooks.* We stand on the shoulders of giants. GitHub (https://github.com/) and Kaggle (https://www.kaggle.com/) provide a plethora of free public Jupyter notebooks that you can use directly, modify, or use as inspiration to create your own notebooks. Make good use of them, and if possible, contribute your own notebooks to the community.

It should be noted that this list of tips is empirical, open to further recommendations, and broad enough to be applied to teaching contexts outside of biotechnology. We hope that these tips, along with the official Jupyter Notebook documentation (https://docs.jupyter.org/), will serve as a starting point for teachers of all disciplines interested in Jupyter notebook design. Additionally, we have created a public GitHub repository (https://github.com/rlebron-bioinfo/biotech-notebooks) with samples of open-source Jupyter notebooks to aid in extending their use in education.

4 Conclusion

Interactive notebooks are flexible, user-friendly tools for introducing students of biotechnology and other disciplines who lack a strong background in mathematics and computer science to computational thinking and programming, which are becoming increasingly important in both research and industry. Literate programming tools, particularly Jupyter notebooks, have grown in popularity among programmers and scientists in recent years, and they are also proving to be useful tools for supporting face-to-face and online teaching. There is also an increasing number of online Jupyter notebooks that are designed to aid in the acquisition of new skills, making them an excellent starting point for engaging students in lifelong and autonomous learning, both of which are critical attitudes in an ever-changing job market.

References

1. United Nations, 'Policy Brief: Education during COVID-19 and beyond', 2020. https://unsdg.un.org/resources/policy-brief-education-during-covid-19-and-beyond (accessed Jun. 25, 2022).
2. S. Pokhrel and R. Chhetri, 'A Literature Review on Impact of COVID-19 Pandemic on Teaching and Learning', *High. Educ. Futur.*, vol. 8, no. 1, pp. 133–141, Jan. 2021, https://doi.org/10.1177/2347631120983481.
3. V. Amnouychokanant, S. Boonlue, S. Chuathong, and K. Thamwipat, 'Online Learning Using Block-based Programming to Foster Computational Thinking Abilities during the COVID-19 Pandemic', *Int. J. Emerg. Technol. Learn.*, vol. 16, no. 13 SE-Papers, pp. 227–247, Jul. 2021, https://doi.org/10.3991/ijet.v16i13.22591.
4. F. Al Ghazali, 'Challenges and opportunities of fostering learner autonomy and self-access learning during the Covid-19 pandemic', *SiSal J.*, vol. 11, no. 3, pp. 114–127, Sep. 2020, https://doi.org/10.37237/110302.
5. J. M. Wing, 'Computational thinking',*Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006.
6. International Society for Technology in Education and Computer Science Teachers Association, 'Operational Definition of Computational Thinking for K–12 Education', 2011. https://cdn.iste.org/www-root/Computational_Thinking_Operational_Definition_ISTE.pdf (accessed Jun. 25, 2022).
7. Y. Arencibia-Rodríguez-del-Rey, I. N. Cawanga Cambinda, C. Deco, C. Bender, R. Avello-Martínez, and K. O. Villalba-Condori, 'Developing computational thinking with a module of solved problems', *Comput. Appl. Eng. Educ.*, vol. 29, no. 3, pp. 506–516, 2021, https://doi.org/10.1002/cae.22214.
8. Y. Li *et al.*, 'Computational Thinking Is More about Thinking than Computing', *J. STEM Educ. Res.*, vol. 3, no. 1, pp. 1–18, 2020, https://doi.org/10.1007/s41979-020-00030-2.
9. J. Domènech-Casal, 'Cracking the genetic code: replicating a scientific discovery', *Sci. Sch.*, vol. 36, pp. 47–51, 2016.
10. M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers, 'The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool', in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–11, https://doi.org/10.1145/3173574.3173748.
11. H. Shen, 'Interactive notebooks: Sharing the code', *Nature*, vol. 515, no. 7525, pp. 151–152, 2014, https://doi.org/10.1038/515151a.
12. A. M. Caughman and E. G. Weigel, 'Biology Students' Math and Computer Science Task Values Are Closely Linked', *CBE—Life Sci. Educ.*, vol. 21, no. 3, p. ar43, 2022, https://doi.org/10.1187/cbe.21-07-0180.
13. F. J. Yuste-Lisbona, J. M. Jiménez-Gómez, C. Capel, and R. Lozano, 'Effective Mapping by Sequencing to Isolate Causal Mutations in the Tomato Genome',*Methods Mol. Biol.*, vol. 2264, pp. 89–103, 2021, https://doi.org/10.1007/978-1-0716-1201-9_7.

14. M. Mirdita, K. Schütze, Y. Moriwaki, L. Heo, S. Ovchinnikov, and M. Steinegger, 'ColabFold: making protein folding accessible to all', *Nat. Methods*, vol. 19, no. 6, pp. 679–682, 2022, https://doi.org/10.1038/s41592-022-01488-1.
15. D. Laurillard, *Teaching as a Design Science*, 1st ed. New York: Routledge, 2012.
16. L. A. Barba *et al.*, 'Teaching and learning with Jupyter', 2019. https://jupyter4edu.github.io/jupyter-edu-book/ (accessed Jun. 25, 2022).
17. J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, 'A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks', in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 507–517, https://doi.org/10.1109/MSR.2019.00077.
18. A. Manches, P. E. McKenna, G. Rajendran, and J. Robertson, 'Identifying embodied metaphors for computing education', *Comput. Human Behav.*, vol. 105, p. 105859, 2020, https://doi.org/10.1016/j.chb.2018.12.037.
19. D. Pérez-Marín, R. Hijón-Neira, A. Bacelo, and C. Pizarro, 'Can computational thinking be improved by using a methodology based on metaphors and scratch to teach computer programming to children?', *Comput. Human Behav.*, vol. 105, p. 105849, 2020, https://doi.org/10.1016/j.chb.2018.12.027.
20. E. Pauwels, 'Mind the metaphor', *Nature*, vol. 500, no. 7464, pp. 523–524, 2013, https://doi.org/10.1038/500523a.