The Journal of Open Source Education

Teaching Python package development: A structured course with learning resources and an instructor's guide

Gerit Wagner [1], **Laureen Thurner** [2], **Carlo Tang** [2], **and Stella Ott** [2]

1 Frankfurt School of Finance & Management **2** Otto-Friedrich-Universität Bamberg

DOI: 10.21105/jose.00292

Software

- Review
- Repository
- Archive

Submitted: 09 January 2025
Published: 28 March 2026

Summary

Although there are many open online courses aimed at teaching Python programming, few educational resources focus on the specific skills required for understanding and developing Python packages. Existing materials typically emphasize programming basics, but the development of Python packages, an essential skill for contributing to the open source community requires deeper knowledge of packaging infrastructure, dependency management, development environments, and best coding practices. This gap leaves learners unprepared for the practicalities of structuring packages, managing dependencies, implementing version control, and ensuring code quality.

To address this, we present educational resources designed to teach Python package development. Our materials include a course plan, detailed syllabus, slides, and practice notebooks that cover introductory sessions on Git and Python, an initial session on identifying project topics, a session on best practices, and a code review session. Additionally, we provide a detailed playbook for instructors to guide course delivery. In the course project, students develop Python packages that extend an existing open source system through a plugin architecture. This approach allows them to implement a standalone package from scratch while integrating their work into a functioning ecosystem.

The course materials were developed as part of the "Open Source Project" offered by the Digital Work Lab at Otto-Friedrich-Universität Bamberg. Informed by iterations and refinements over four semesters, these resources provide a well-structured and engaging learning experience. The course repository consists of a landing page, featuring course pages, instructor notes, slides, and practice notebooks. A preconfigured development setup is offered for GitHub Codespaces, ensuring that the required development environment is preinstalled. The course materials are hosted on GitHub and available for reuse and adaptation.

Statement of need

A broad range of Python learning resources is accessible online, reflecting its prominence as a programming language across industries. Massive Open Online Courses (MOOCs), such as those offered by Coursera and edX, cater to vast audiences with substantial enrollment figures. For instance, Harvard's CS50's Introduction to Programming with Python and IBM's Python for Data Science, AI and Development highlight beginner-friendly content, emphasizing foundational skills like using libraries. These courses are frequently structured around paid certificates and follow conventional formats, making them popular among learners seeking basic programming knowledge or career-oriented credentials. However, these MOOCs rarely delve into more advanced topics like Python package development, leaving a noticeable gap for learners aiming to contribute to the open source ecosystem.

In contrast, resources available for teaching Python package development primarily target self-learners. Materials like those offered by PyOpenSci or the book of Beuzen & Timbers (2020) provide valuable insights into creating reusable, distributable Python libraries. However, these materials often lack the structured, interactive learning experience offered by formal courses. Consequently, while existing resources equip individual learners with practical tools for package development, they do not cater to a broader audience. Addressing this gap requires tailored educational materials that combine accessibility with the depth necessary to teach Python package development.

Table 1: Overview of selected Python courses

Course title (provider)	Target	Duration	Libraries
Python for Data Science, AI and Development (IBM, Coursera)	Beginner	25h	Using libraries
Python for Everybody Specialization (University of Michigan, Coursera)	Beginner	2 months at 10h/week	
CS50's Introduction to Programming with Python (Harvard University, edX)	Beginner	10 weeks	Using libraries
Introduction to Computer Science and Programming Using Python (MIT, edX)	Beginner	9 weeks	
CS50's Introduction to Artificial Intelligence with Python (Harvard University, edX)	Beginner	7 weeks	Using libraries
Machine Learning with Python: A Practical Introduction (IBM, edX)	Beginner	5 weeks	Using libraries
Programming for Everybody: Getting Started with Python (University of Michigan, edX)	Beginner	7 weeks	
Applied Data Science with Python Specialization (University of Michigan, Coursera)	Intermediate	4 months at 10h/week	Using libraries
CS50's Web Programming with Python and JavaScript (Harvard University, edX)	Intermediate	12 weeks	

The overview of selected Python courses in Table 1 illustrates the popularity and scope of beginner-friendly MOOCs, and highlights the gap in resources for advanced Python package development.

Generally, Python package development can be helpful for a range of purposes:

1. **Reusability** Writing Python code from scratch is time-consuming and error-prone. Many tasks, especially in fields like data science, web development, and automation, have well-established solutions in existing Python packages. Learning how to develop packages enables students to make existing code available for reuse, and it also develops understanding and skills related to the use of existing packages.
2. **Access to specialized functionality** Considering that the Python core only includes general-purpose built-in functionality, packages are often required to provide specialized functionality. For instance, this includes tasks like machine learning (TensorFlow, Scikit-learn), scientific computing (SciPy), or web development (Flask, Django). Understanding these packages allows students to access to a wide range of tools and resources that extend Python's functionality for specific purposes.
3. **Dependency management** Python packages often rely on external libraries that are updated over time to introduce new features or address security vulnerabilities. Managing these dependencies effectively is an important skill, as different packages may require specific versions of the same library, leading to potential conflicts. Tools like pip and virtual environments provide mechanisms for isolating dependencies, but ensuring stability and reproducibility requires a more comprehensive approach. One element are cross-platform and cross-Python-version testing strategies to verify that a Python package functions consistently across different environments.
4. **Version control, collaborative development, and open source contribution** Version control systems, such as Git, are used to manage changes in Python package development, allowing developers to track modifications, revert to previous states,

and maintain a clear project history. Git also supports collaborative development by enabling multiple contributors to work on the same codebase while managing conflicts and integrating changes. Platforms such as GitHub or GitLab facilitate these workflows through practices like pull requests, code reviews, and issue tracking. By participating in such environments, students gain experience contributing to shared projects and engaging in open source communities, where their work can be reused, improved, and extended by others. Ideally, this exposure may foster an appreciation for collaborative coding and emphasizes the importance of building packages that are maintainable, accessible, and aligned with community standards.

5. **Scalability and maintainability of projects** As a project grows in complexity, managing code becomes difficult without proper structure. Packages help modularize code, separating it into manageable units, and using continuous integration tools to maintain code quality. Understanding package development ensures that code is scalable and maintainable. This is essential when building large-scale applications where different parts of the software can be independently developed, tested, and maintained.

Our target audience consists of Bachelor's students with foundational programming skills.

Learning objectives and outline

The specific learning objectives for the capstone project are:

1. **Understand the structure and distribution of Python packages**
   Students will learn how to design, organize, and structure a Python package according to best practices, including creating modular code, setting up essential files (e.g., `pyproject.toml`), and distributing the package using PyPI.
2. **Apply version control and manage dependencies**
   Students will develop skills in managing package dependencies and versioning, using tools like virtualenv and poetry for isolated environments, and ensuring compatibility across various project setups. This includes understanding the role of code quality tools, the importance of semantic versioning, and maintaining stable software releases.
3. **Collaboratively develop a Python package**
   Students will gain hands-on experience contributing to open source Python projects by collaborating on GitHub, creating pull requests, resolving issues, and following community-driven development standards. They will also learn how to write documentation and test their packages to ensure quality and usability.

These learning objectives are achieved through the development of small Python packages that extend an existing open source Package through a plugin architecture.

Figure 1 provides an overview of the course timeline, showcasing sessions and group work activities that facilitate a step-by-step progression through Python package development concepts. The timeline emphasizes iterative learning, with early sessions focused on foundational skills, followed by group work phases that foster collaboration and practical application. During this time, students are encouraged to iterate between individual coding, group sessions, and hacking sessions with the instructor to discuss current challenges and next steps.

In addition, a *Best Practices* session is offered at the beginning of the group work phase. Toward the end of the course, students will open a pull request with their work and participate in a code review session in which they adopt the perspective of a maintainer and evaluate the code of another group. Code improvements are implemented within a week, and student reflections are discussed at the end. When merging the contributions, we include students as contributors of the package.

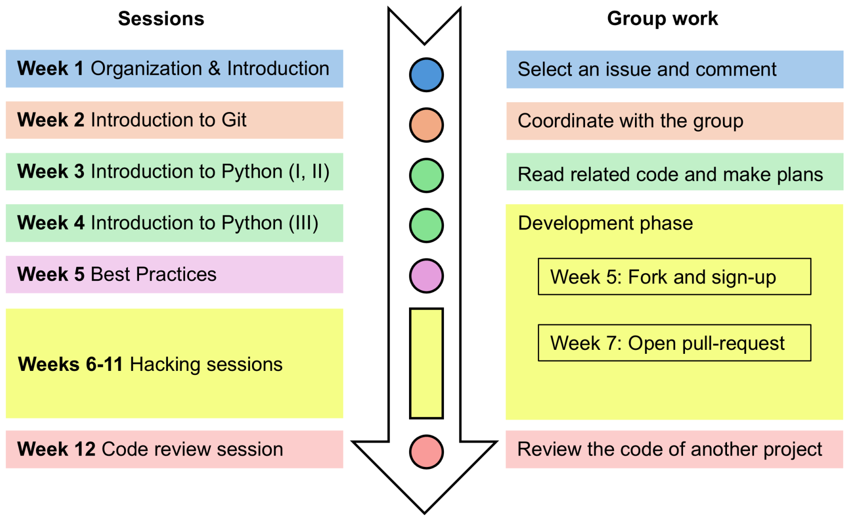
Sessions
Group work
Week 1 Organization & Introduction
Week 2 Introduction to Git
Week 3 Introduction to Python (I, II)
Week 4 Introduction to Python (III)
Week 5 Best Practices
Weeks 6-11 Hacking sessions
Week 12 Code review session
Select an issue and comment
Coordinate with the group
Read related code and make plans
Development phase
Week 5: Fork and sign-up
Week 7: Open pull-request
Review the code of another project

Figure 1: Course timeline with sessions and group work activities

Teaching materials

The delivery of the course is designed to foster active learning through a blend of in-person sessions, individual group work, and interactive hacking sessions, all facilitated by the instructor. This approach emphasizes collaboration and hands-on practice, enabling students to engage deeply with Python package development in a supportive environment. The materials are structured for complex and integrated learning activities, with a strong focus on Git-based collaboration. Students work together on shared repositories, navigating real-world workflows such as branching, merging, and resolving conflicts, which mirror professional development environments. The Git component of the course builds on the work of Wagner & Thurner (2025) to ensure a robust foundation in version control while emphasizing practical applications that enhance both technical skills and teamwork.

Table 2: Materials

Resource	Description and focus
Landing page	Provides an accessible overview of the course, aimed to engage students.
Syllabus	Offers a structured, detailed overview of course objectives, content, and pedagogical approach, complementing the landing page with broader course context.
Slides	Slides in Markdown (Marp) format.
Notebooks	Designed to engage directly with the Python package, using `git reset` to access solutions, supporting a smooth, practical learning flow.
Teaching notes	The teaching notes contain preparation checklists, scheduled mailings, session readers, and a concept.

Pedagogical considerations

There are distinct benefits to focusing on the development of Python packages that serve as plugins to a preexisting project[1]. It allows students to understand how packages are built from scratch while simultaneously appreciating the features of a fully developed Python package. Developing a plugin requires students to implement the essential elements of a Python package—such as creating a project structure, configuring the `pyproject.toml`, and managing dependencies—while integrating the package into an existing project that already provides meaningful functionality. This helps students connect packaging fundamentals with a working application and see how their code interacts with a larger package. At the same time, plugin development introduces architectural concepts that are central to realistic Python packages and modern software ecosystems, including modularity, interfaces, and extensibility. Finally, by extending an existing project, students engage in a realistic development environment that provides a scaffold of best practices related to documentation, version control workflows, and issue discussions. This creates a more authentic development experience than building isolated examples from scratch and increases the likelihood that student-developed extensions could eventually be shared or reused within the ecosystem.

Teaching Python package development requires a structured approach that balances simplicity with depth, ensuring students build a solid foundation before progressing to more advanced topics. The course design is informed by key pedagogical principles:

1. **Select and simplify**
   To reduce cognitive overload, we prioritize simplicity in tools and workflows. For example, we use GitHub Codespaces to standardize setups, eliminating issues related to different operating systems and environment configurations. A focused approach aligns with cognitive load theory (Sweller, 1994), helping students concentrate on core concepts.

2. **Gradually progress in complexity**
   Starting with basic Python and Git skills, the material builds incrementally, introducing concepts like dependency management and package distribution after students have developed sufficient familiarity with foundational skills. This approach reduces the risk of overwhelming learners (Anderson et al., 2001).

3. **Learn interactively and in groups**
   Interactive and collaborative learning plays a crucial role in student engagement and knowledge retention (Guzdial, 2001). The course incorporates live coding sessions, and group-based exercises to make practices as accessible as possible and encourage active participation (Vial & Negoita, 2018). We build on the principles of active learning to promote deeper understanding through hands-on practice and peer collaboration. In particular, group problem-solving can foster a collaborative environment where learners can exchange ideas, learn from one another, and build confidence in their coding skills (Freeman et al., 2014).

Development environment

Our recommended setup for Python package development is GitHub Codespaces, a cloud-based development solution featuring a graphical interface of VisualStudio Code as well as a full Python environment with preinstalled dependencies and configuration[2]. With Codespaces, students can start their work directly from a browser, where all

[1]We thank @NickleDave for raising this point during the review process and encouraging us to reflect more carefully on the benefits of this particular setting.

[2]Local VirtualBox images were too slow on most student machines, and resources for self-hosted virtual machines were not available.

necessary dependencies are automatically configured. GitHub Codespaces offers several key advantages for Python package development. Offloading all computational tasks to remote servers eliminates the performance issues that often arise when running development environments on local machines. Additionally, the environment is fully standardized, meaning every student works with the same configuration, reducing the variability and potential issues seen in local setups. The Codespaces startup scripts effectively allow us to set up the development environment automatically and without user interaction. This approach not only saves time but also mirrors the benefits described by Malan (2024), where containerization minimized technical challenges and enhanced the learning experience.

For students who prefer a local development setup, we also offer options like Windows Subsystem for Linux (WSL) for Windows users, ensuring they can work with a Linux-like environment while still on their native operating system. This provides flexibility while maintaining the core benefits of a standardized development environment. By enabling students to work in a consistent environment regardless of their operating system, we ensure that everyone has access to the same tools, configurations, and workflows. It also ensures equal opportunities, regardless of students' choice of operating system and enables them to collaborate effectively without technical barriers.

Reuse and modification of materials

The materials provided in this course are designed for easy reuse and modification by other instructors. While the course uses the CoLRev Python package (Wagner & Prester, 2024) as the example context, the materials allow instructors to adapt the content to focus on different Python packages. The learning environment, hosted on GitHub and built with the Just-the-Docs framework, can be cloned, enabling instructors to replicate the entire setup. All instructional content, including slides and practice notebooks, are automatically generated and updated via GitHub Actions, ensuring the materials remain up-to-date.

Story of the project

This project was developed at Otto-Friedrich-Universität Bamberg, where Gerit offered a team-based course to provide students with hands-on experiences in programming and software development. While other courses in the program had a stronger focus on management topics, this project was designed to help students engage with realistic open source software engineering practices. The goal was to move beyond short, standalone Python scripts, and contribute collaboratively to public, functional, maintainable, and open source software packages.

Building on Gerit's role as lead developer of the CoLRev open source environment, the course adopted CoLRev as a reference project. Students developed small standalone Python packages that extend the system through a plugin architecture, allowing them to reuse existing functionality (e.g., for loading bibliographic data or interacting with APIs) and extend it by implementing new features such as API wrappers or automation tools. This setup provided an authentic context for learning how to organize projects in teams using Git and GitHub, manage dependencies, write tests, and produce corresponding documentation.

Over time, the course evolved substantially through iterative refinement based on student feedback. Early cohorts found the technical scope demanding, prompting the integration of more scaffolding, structured guidance, and formative feedback. Since then, the course has grown into a well-established project with high student satisfaction and recommendation rates. Five cohorts of students have contributed to CoLRev and related open source packages, many continuing their engagement through bachelor's theses on extensions such as the search-query (Eckhardt et al., 2026) or BibDedupe (Wagner, 2024) packages.

References

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J., & Wittrock, M. C. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives, complete edition*. Longman Publishing Group.

Beuzen, T., & Timbers, T. (2020). *Python packages* (1st ed.). Chapman & Hall/CRC The Python Series. ISBN: 9781138332250

Eckhardt, P., Ernst, K. M., Fleischmann, T., Geßler, A., Schnickmann, K., Thurner, L., & Wagner, G. (2026). search-query: An open source Python library for academic search queries. *Journal of Open Source Software*, *11*(118), 8775. https://doi.org/10.21105/JOSS.08775

Freeman, S., Eddy, S. L., McDonough, M., Smith, M. K., Okoroafor, N., Jordt, H., & Wenderoth, M. P. (2014). Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, *111*(23), 8410–8415. https://doi.org/10.1073/PNAS.1319030111

Guzdial, M. (2001). Use of collaborative multimedia in computer science classes. *ACM SIGCSE Bulletin*, *33*(3), 17–20. https://doi.org/10.1145/507758.377452

Malan, D. J. (2024). Containerizing CS50: Standardizing students' programming environments. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (pp. 534–540). https://doi.org/10.1145/3649217.3653567

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, *4*(4), 295–312. https://doi.org/10.1016/0959-4752(94)90003-5

Vial, G., & Negoita, B. (2018). Teaching programming to non-programmers - the case of Python and Jupyter Notebooks. *Proceedings of the International Conference on Information Systems*.

Wagner, G. (2024). BibDedupe: An open-source Python library for deduplication of bibliographic records. *Journal of Open Source Software*, *9*(97), 6318. https://doi.org/10.21105/joss.06318

Wagner, G., & Prester, J. (2024). *CoLRev: An open-source environment for collaborative reviews* (Version 0.16.2). https://doi.org/10.5281/ZENODO.11668338

Wagner, G., & Thurner, L. (2025). Teaching tip rethinking how we teach Git: Pedagogical recommendations and practical strategies for the information systems curriculum. *Journal of Information Systems Education*, *36*(1), 1–12. https://doi.org/10.62273/BTKM5634